Pilote de CESAR

Documentation utilisateur

Philippe LEVEQUE
philippe.leveque@univ-eiffel.fr

Version 2022_05_18

Table des matières

1 Introduction **1**
1.1 Contexte . . . 1
1.2 Vue d'ensemble . . . 2

2 Langage de commandes Pilote **5**
2.1 Principe général de mise en données . . . 6
2.2 Lecture et écriture sur fichier . . . 7
2.2.1 Classe FICHIER . . . 7
2.3 Affectation de propriétés . . . 8
2.3.1 Classe AFFECT . . . 8
2.3.2 Classe PROPRIETE . . . 8
2.3.3 Classe LIEU . . . 9
2.4 Extraction de grandeurs . . . 10
2.4.1 Classe EXTRACT . . . 10
2.4.2 Classe GRANDEUR . . . 10
2.5 Définition d'une étude . . . 10
2.5.1 Classe ETUDE . . . 10
2.5.2 Classe PHASAGE . . . 12
2.6 Génération d'un maillage à partir d'un domaine . . . 12
2.6.1 Classe DOMAINE . . . 12
2.6.2 Classe OPERATION . . . 13
2.7 Données relatives aux domaines . . . 14
2.7.1 Classe POINT . . . 14
2.7.2 Classe LIGNE . . . 14
2.7.3 Classe CERCLE . . . 15
2.7.4 Classe ELLIPSE . . . 15
2.7.5 Classe SPLINE . . . 15
2.7.6 Classe BSPLINE . . . 16
2.7.7 Classe BOUCLE . . . 16
2.7.8 Classe SURFACE . . . 16
2.7.9 Classe VOLUME . . . 17
2.7.10 Classe ASSEMBLAGE . . . 17
2.7.11 Classe BLOC . . . 17
2.7.12 Classe EXTRUSION . . . 18
2.7.13 Classe TRANSFORMATION . . . 18
2.7.14 Classe STRUCTURE . . . 19
2.7.15 Classe RECOMBINE . . . 20
2.7.16 Classe COHERENCE . . . 20
2.7.17 Classe FINESSE . . . 20

2.7.18 Classe DESTRUCTION . . . 21
2.7.19 Classe OPTION . . . 21
2.7.20 Classe ENTITES . . . 21
2.7.21 Classe ID . . . 22
2.8 Données relatives aux maillages . . . 23
2.8.1 Classe MAILLAGE . . . 23
2.8.2 Classe GMESH . . . 27
2.8.3 Classe GROUPE . . . 27
2.8.4 Classe COOR_2D . . . 29
2.8.5 Classe COOR_3D . . . 30
2.9 Données relatives aux modèles . . . 30
2.9.1 Classe MODELE . . . 30
2.9.2 Classe FORMUL_MECA . . . 31
2.9.3 Classe FORMUL_DIFFUS . . . 32
2.9.4 Classe FORMUL_COUPLAG . . . 32
2.10 Données relatives aux matériaux . . . 32
2.10.1 Classe MATERIAU . . . 32
2.10.2 Classe ELAS_LINE_ISO . . . 33
2.10.3 Classe ELAS_LINE_ORTHO . . . 34
2.10.4 Classe BETON_JEUNE_AGE . . . 34
2.10.5 Classe MOHR_COUL_SANS_ECROU . . . 35
2.10.6 Classe VON_MISES_SANS_ECROU . . . 35
2.10.7 Classe VON_MISES_AVEC_ECROU . . . 36
2.10.8 Classe DRUCK_PRAG_SANS_ECROU . . . 36
2.10.9 Classe DRUCK_PRAG_AVEC_ECROU . . . 36
2.10.10 Classe CRIT_PARABOL . . . 37
2.10.11 Classe VERMEER . . . 37
2.10.12 Classe CAM_CLAY_MODIF . . . 38
2.10.13 Classe PREVOST_HOEG . . . 39
2.10.14 Classe CRIT_ORIENT . . . 39
2.10.15 Classe HOEK_BROWN . . . 40
2.10.16 Classe TRESCA_ANISO . . . 40
2.10.17 Classe ELAS_DILAT_ISO . . . 41
2.10.18 Classe ASTER_ELAS . . . 41
2.10.19 Classe ASTER_ELAS_ORTH . . . 42
2.10.20 Classe ASTER_VMIS_ISOT_LINE . . . 42
2.10.21 Classe MATER_RENFORCE . . . 43
2.10.22 Classe ELASTO_PLAST_1D . . . 44
2.10.23 Classe COMPOSITE . . . 44
2.10.24 Classe DIFFUS_LINE_2D . . . 45
2.10.25 Classe DIFFUS_LINE_3D . . . 45
2.10.26 Classe ECHANG_LINE . . . 46
2.11 Données relatives aux caractéristiques . . . 46
2.11.1 Classe CARACTERISTIQUE . . . 46
2.11.2 Classe CONTR_PLANE . . . 47
2.11.3 Classe DEFOR_PLANE . . . 48
2.11.4 Classe AXISYM . . . 48
2.11.5 Classe MASSIF_3D . . . 49
2.11.6 Classe BAR_2D . . . 51

2.11.7 Classe BAR_3D . . . 51
2.11.8 Classe BAR_FROT_LINE . . . 52
2.11.9 Classe BAR_FROT_PLAS . . . 52
2.11.10 Classe BAR_FROT_BILINE . . . 52
2.11.11 Classe BAR_FROT_RACINE . . . 53
2.11.12 Classe POUT_2D . . . 53
2.11.13 Classe POUT_3D . . . 54
2.11.14 Classe COQUE . . . 54
2.11.15 Classe POUT_3D_MULTI . . . 55
2.11.16 Classe COQUE_MULTI . . . 56
2.12 Données relatives aux activations . . . 56
2.12.1 Classe ACTIVATION . . . 56
2.13 Données relatives aux liaisons . . . 57
2.13.1 Classe LIAISON . . . 57
2.13.2 Classe RELA_LINE . . . 58
2.13.3 Classe MATR_ELEM . . . 59
2.13.4 Classe CONTACT . . . 60
2.13.5 Classe JOINT . . . 61
2.14 Données relatives aux conditions limites . . . 62
2.14.1 Classe COND_LIMITE . . . 62
2.14.2 Classe DDL_IMPOSE . . . 63
2.15 Données relatives aux chargements . . . 64
2.15.1 Classe CAS_CHARGEMENT . . . 64
2.15.2 Classe CHARGEMENT . . . 64
2.15.3 Classe FORC_NOEU . . . 66
2.15.4 Classe FORC_ARETE_2D . . . 66
2.15.5 Classe PRESS_ARETE_2D . . . 66
2.15.6 Classe CISAIL_ARETE_2D . . . 67
2.15.7 Classe PRESS_HYDRO_2D . . . 67
2.15.8 Classe FLUX_ECHANG_LINE_2D . . . 67
2.15.9 Classe FLUX_ECHANG_LINE_3D . . . 67
2.15.10 Classe EXCAV_2D . . . 68
2.15.11 Classe EXCAV_3D . . . 68
2.15.12 Classe DECONFIN_2D . . . 68
2.15.13 Classe DECONFIN_3D . . . 69
2.15.14 Classe FORC_FACE_3D . . . 70
2.15.15 Classe PRESS_FACE_3D . . . 70
2.15.16 Classe PRESS_HYDRO_3D . . . 70
2.15.17 Classe POIDS . . . 71
2.15.18 Classe FORC_MAILLE_COQ . . . 71
2.15.19 Classe FORC_MAILLE_POUT_2D . . . 71
2.15.20 Classe FORC_MAILLE_POUT_3D . . . 72
2.15.21 Classe CONTR_UNIF . . . 72
2.15.22 Classe CONTR_GEOSTAT . . . 73
2.16 Données relatives aux analyses . . . 74
2.16.1 Classe STAT_LINE . . . 74
2.16.2 Classe STAT_NON_LINE . . . 74
2.16.3 Classe MODAL_LINE . . . 77
2.16.4 Classe DYNA_LINE_TEMP . . . 78

2.16.5 Classe DYNA_LINE_HARMO . . . 79
2.16.6 Classe THERM_BETON_JEUNE_AGE . . . 80
2.16.7 Classe MECA_BETON_JEUNE_AGE . . . 81
2.17 Données relatives aux post-traitements . . . 81
2.17.1 Classe POST_TRAITEMENT . . . 81
2.18 Données relatives aux résolutions . . . 82
2.18.1 Classe RESOLUTION . . . 82
2.18.2 Classe PHASE . . . 83
2.19 Données relatives aux résultats . . . 84
2.19.1 Classe RESULTAT . . . 84
2.19.2 Classe DEPLA_2D . . . 87
2.19.3 Classe DEPLA_3D . . . 88
2.19.4 Classe DEPLA_2D_ROTA . . . 88
2.19.5 Classe DEPLA_3D_ROTA . . . 89
2.19.6 Classe ACCEL_2D . . . 89
2.19.7 Classe ACCEL_3D . . . 89
2.19.8 Classe CONTR_2D . . . 90
2.19.9 Classe CONTR_3D . . . 90
2.19.10 Classe DEFORM_TOTAL_2D . . . 91
2.19.11 Classe DEFORM_TOTAL_3D . . . 91
2.19.12 Classe REAC_2D . . . 92
2.19.13 Classe REAC_3D . . . 92
2.19.14 Classe EFFORT_BAR_2D . . . 93
2.19.15 Classe EFFORT_BAR_3D . . . 93
2.19.16 Classe EFFORT_POUTR_2D . . . 93
2.19.17 Classe EFFORT_POUTR_3D . . . 93
2.19.18 Classe TEMP . . . 94
2.19.19 Classe GRAD_TEMP_2D . . . 94
2.19.20 Classe GRAD_TEMP_3D . . . 95
2.19.21 Classe FIELD_ . . . 95
2.20 Exemple d'étude . . . 95
2.20.1 Script . . . 95
2.20.2 Commentaires . . . 97

3 Langage de commandes CESAR 103
3.1 Principe général de mise en données . . . 104
3.2 Définition d'un jeu de données . . . 104
3.2.1 Classe JDD . . . 104
3.3 Données utilitaires . . . 106
3.3.1 Classe TEST . . . 106
3.3.2 Classe EXEC . . . 107
3.3.3 Classe ILIG . . . 107
3.3.4 Classe COMT . . . 107
3.3.5 Classe EXPO . . . 108
3.3.6 Classe GEFI . . . 108
3.3.7 Classe IMPR . . . 108
3.4 Données relatives aux noeuds . . . 110
3.4.1 Classe COOR . . . 110
3.5 Données relatives aux éléments . . . 110

3.5.1 Classe ELEM 110
3.6 Données relatives au modèle de mécanique bidimensionnelle 112
3.6.1 Classe MB 112
3.6.2 Classe MB_ELI 114
3.6.3 Classe MB_ELO 114
3.6.4 Classe MB_BJA 115
3.6.5 Classe MB_MCSE 115
3.6.6 Classe MB_VMSE 116
3.6.7 Classe MB_VMAE 117
3.6.8 Classe MB_DPSE 117
3.6.9 Classe MB_DPAE 118
3.6.10 Classe MB_CP 118
3.6.11 Classe MB_VE 119
3.6.12 Classe MB_NO 120
3.6.13 Classe MB_CCM 121
3.6.14 Classe MB_PH 121
3.6.15 Classe MB_CO 122
3.6.16 Classe MB_HB 122
3.6.17 Classe MB_ME 123
3.6.18 Classe MB_MCSE_EO 124
3.6.19 Classe MB_TA 125
3.6.20 Classe MB_RBS 126
3.6.21 Classe MB_WWS 127
3.6.22 Classe MB_WWM 128
3.6.23 Classe MB_EDI 129
3.6.24 Classe MB_BAO 130
3.6.25 Classe CMP_ELAS 130
3.6.26 Classe ELAS_LI 131
3.6.27 Classe ELAS_LIV 131
3.6.28 Classe ELAS_FC 131
3.6.29 Classe ELAS_HSM 132
3.6.30 Classe ELAS_ANL 132
3.6.31 Classe CMP_CRT 132
3.6.32 Classe FNC_T 133
3.6.33 Classe FNC_VM 133
3.6.34 Classe FNC_C 133
3.6.35 Classe FNC_MC 133
3.6.36 Classe FNC_MCV 134
3.6.37 Classe FNC_DP 134
3.6.38 Classe FNC_CCM 134
3.6.39 Classe FNC_CO 134
3.6.40 Classe FNC_VMC 135
3.6.41 Classe FNC_DPC 135
3.6.42 Classe FNC_HSM 135
3.6.43 Classe CMP_POT 135
3.6.44 Classe CMP_ECR 136
3.6.45 Classe ECR_L 137
3.6.46 Classe ECR_PH 137
3.6.47 Classe ECR_PHM 137

3.6.48 Classe ECR_NMC . . . 137
3.6.49 Classe ECR_VM . . . 137
3.6.50 Classe ECR_VMV . . . 138
3.6.51 Classe ECR_V . . . 138
3.6.52 Classe ECR_VD . . . 138
3.6.53 Classe ECR_VPP . . . 138
3.6.54 Classe ECR_VRS . . . 139
3.6.55 Classe ECR_CH . . . 139
3.6.56 Classe ECR_HSM . . . 139
3.6.57 Classe CMP_RENF . . . 139
3.6.58 Classe RENF_EL . . . 140
3.6.59 Classe RENF_PP . . . 140
3.6.60 Classe RENF_H . . . 140
3.6.61 Classe RENF_R . . . 140
3.6.62 Classe RENF_DC . . . 141
3.6.63 Classe RENF_DS . . . 141
3.7 Données relatives au modèle de mécanique tridimensionnelle . . . 141
3.7.1 Classe MT . . . 141
3.7.2 Classe MT_ELI . . . 143
3.7.3 Classe MT_ELO . . . 143
3.7.4 Classe MT_BJA . . . 144
3.7.5 Classe MT_MCSE . . . 144
3.7.6 Classe MT_VMSE . . . 145
3.7.7 Classe MT_VMAE . . . 145
3.7.8 Classe MT_DPSE . . . 145
3.7.9 Classe MT_DPAE . . . 146
3.7.10 Classe MT_CP . . . 146
3.7.11 Classe MT_VE . . . 147
3.7.12 Classe MT_NO . . . 147
3.7.13 Classe MT_CCM . . . 148
3.7.14 Classe MT_PH . . . 148
3.7.15 Classe MT_CO . . . 149
3.7.16 Classe MT_HB . . . 149
3.7.17 Classe MT_RBS . . . 150
3.7.18 Classe MT_WWS . . . 152
3.7.19 Classe MT_WWM . . . 152
3.7.20 Classe MT_EDI . . . 153
3.7.21 Classe MT_BAO . . . 153
3.8 Données relatives au modèle de poutre bidimensionnelle . . . 154
3.8.1 Classe PB . . . 154
3.8.2 Classe PB_ELI . . . 154
3.9 Données relatives au modèle de poutre tridimensionnelle . . . 155
3.9.1 Classe PT . . . 155
3.9.2 Classe PT_STD_ELI . . . 155
3.9.3 Classe PT_MF . . . 156
3.9.4 Classe PT_LF . . . 157
3.9.5 Classe PT_LF_ELI . . . 157
3.9.6 Classe PT_LF_VMSE . . . 158
3.9.7 Classe PT_LF_VMAE . . . 158

3.9.8 Classe PT_LF_CP . . . 159
3.9.9 Classe PT_LF_WWS . . . 159
3.9.10 Classe PT_LF_WWM . . . 160
3.9.11 Classe PT_CF . . . 160
3.10 Données relatives au modèle de coque . . . 161
3.10.1 Classe CO . . . 161
3.10.2 Classe CO_STD_ELI . . . 161
3.10.3 Classe CO_MC . . . 162
3.10.4 Classe CO_LC . . . 162
3.10.5 Classe CO_LC_ELI . . . 163
3.10.6 Classe CO_LC_VMSE . . . 163
3.10.7 Classe CO_LC_VMAE . . . 163
3.10.8 Classe CO_LC_CP . . . 164
3.10.9 Classe CO_LC_WWS . . . 164
3.10.10 Classe CO_LC_WWM . . . 165
3.11 Données relatives au modèle de contact . . . 165
3.11.1 Classe FD . . . 165
3.11.2 Classe FD_AD . . . 166
3.11.3 Classe FD_FC . . . 166
3.11.4 Classe FD_GP . . . 167
3.12 Données relatives au modèle de joint . . . 168
3.12.1 Classe EJ . . . 168
3.12.2 Classe EJ_AD . . . 168
3.12.3 Classe EJ_FC . . . 168
3.12.4 Classe EJ_GP . . . 169
3.12.5 Classe EJ_EL . . . 169
3.12.6 Classe EJ_EPP . . . 169
3.13 Données relatives au modèle de barre bidimensionnelle . . . 170
3.13.1 Classe BB . . . 170
3.13.2 Classe BB_ELI . . . 170
3.14 Données relatives au modèle de barre tridimensionnelle . . . 170
3.14.1 Classe BT . . . 170
3.14.2 Classe BT_ELI . . . 171
3.15 Données relatives au modèle de barre frottante . . . 171
3.15.1 Classe KR . . . 171
3.15.2 Classe KR_ELI . . . 172
3.15.3 Classe KR_INT_EL . . . 172
3.15.4 Classe KR_INT_ELP . . . 173
3.15.5 Classe KR_INT_EBP . . . 173
3.15.6 Classe KR_INT_ERP . . . 173
3.16 Données relatives aux relations linéaires . . . 174
3.16.1 Classe RL . . . 174
3.17 Données relatives aux éléments spéciaux . . . 174
3.17.1 Classe SP . . . 174
3.18 Données relatives au modèle de mécanique bidimensionnelle axisymétrique . . . 175
3.18.1 Classe AX . . . 175
3.18.2 Classe AX_ELI . . . 175
3.19 Données relatives au modèle de diffusion bidimensionnelle . . . 176
3.19.1 Classe DB . . . 176

3.19.2 Classe DB_CCH . . . 176
3.19.3 Classe DB_EMP . . . 177
3.19.4 Classe DB_NLG . . . 177
3.19.5 Classe DB_CP . . . 178
3.19.6 Classe DB_CPP . . . 178
3.20 Données relatives au modèle de diffusion tridimensionnelle . . . 179
3.20.1 Classe DT . . . 179
3.20.2 Classe DT_CCH . . . 180
3.20.3 Classe DT_EMP . . . 180
3.20.4 Classe DT_NLG . . . 181
3.20.5 Classe DT_CP . . . 181
3.20.6 Classe DT_CPP . . . 182
3.21 Données relatives au modèle d'échange bidimensionnel . . . 183
3.21.1 Classe EB . . . 183
3.21.2 Classe EB_CCH . . . 184
3.21.3 Classe EB_EMP . . . 184
3.21.4 Classe EB_NLG . . . 184
3.22 Données relatives au modèle d'échange tridimensionnel . . . 185
3.22.1 Classe ET . . . 185
3.22.2 Classe ET_CCH . . . 185
3.22.3 Classe ET_EMP . . . 185
3.22.4 Classe ET_NLG . . . 186
3.23 Données relatives au modèle bidimensionnel pour la recherche d'une surface libre 186
3.23.1 Classe SB . . . 186
3.24 Données relatives au modèle bidimensionnel de thermo-mécanique des milieux poreux . . . 187
3.24.1 Classe OB . . . 187
3.24.2 Classe OB_ELI . . . 188
3.24.3 Classe OB_ELO . . . 189
3.24.4 Classe OB_MC . . . 190
3.24.5 Classe OB_VMSE . . . 190
3.24.6 Classe OB_VMAE . . . 191
3.24.7 Classe OB_DPSE . . . 191
3.24.8 Classe OB_DPAE . . . 192
3.24.9 Classe OB_CP . . . 192
3.24.10 Classe OB_VE . . . 192
3.24.11 Classe OB_NO . . . 193
3.24.12 Classe OB_CCM . . . 194
3.24.13 Classe OB_PH . . . 194
3.24.14 Classe OB_CO . . . 194
3.25 Données relatives au modèle tridimensionnel de thermo-mécanique des milieux poreux . . . 195
3.25.1 Classe OT . . . 195
3.25.2 Classe OT_ELI . . . 196
3.25.3 Classe OT_ELO . . . 197
3.25.4 Classe OT_VMSE . . . 199
3.25.5 Classe OT_VMAE . . . 199
3.25.6 Classe OT_DPSE . . . 199
3.25.7 Classe OT_DPAE . . . 200

3.25.8 Classe OT_CP . . . 200
3.25.9 Classe OT_VE . . . 201
3.25.10 Classe OT_NO . . . 201
3.25.11 Classe OT_CCM . . . 202
3.25.12 Classe OT_PH . . . 202
3.25.13 Classe OT_CO . . . 203
3.26 Données relatives aux conditions limites . . . 203
3.26.1 Classe COND . . . 203
3.26.2 Classe COND_DDL . . . 204
3.26.3 Classe COND_IMP . . . 204
3.26.4 Classe COND_NUL . . . 205
3.26.5 Classe COND_REP . . . 206
3.27 Données relatives aux chargements . . . 207
3.27.1 Classe CHAR . . . 207
3.27.2 Classe CHAR_CNU . . . 208
3.27.3 Classe CHAR_CUR . . . 208
3.27.4 Classe CHAR_DVU . . . 208
3.27.5 Classe CHAR_ECH . . . 209
3.27.6 Classe CHAR_EFD . . . 209
3.27.7 Classe CHAR_FNU . . . 210
3.27.8 Classe CHAR_FOS . . . 211
3.27.9 Classe CHAR_FUR . . . 211
3.27.10 Classe CHAR_LAM . . . 211
3.27.11 Classe CHAR_DEC . . . 213
3.27.12 Classe CHAR_PHS . . . 214
3.27.13 Classe CHAR_PNU . . . 214
3.27.14 Classe CHAR_PNC . . . 215
3.27.15 Classe CHAR_POI . . . 216
3.27.16 Classe CHAR_PUR . . . 216
3.27.17 Classe CHAR_SIG . . . 217
3.27.18 Classe CHAR_SOL . . . 219
3.28 Données relatives au type de calcul . . . 220
3.28.1 Classe AXIF . . . 220
3.28.2 Classe DTLI . . . 222
3.28.3 Classe DTNL . . . 222
3.28.4 Classe DYNI . . . 224
3.28.5 Classe FLAM . . . 224
3.28.6 Classe LINC . . . 225
3.28.7 Classe LINE . . . 225
3.28.8 Classe LINH . . . 226
3.28.9 Classe MCNL . . . 226
3.28.10 Classe MEXO . . . 228
3.28.11 Classe MODE . . . 229
3.28.12 Classe MPLI . . . 229
3.28.13 Classe MPNL . . . 230
3.28.14 Classe NSAT . . . 232
3.28.15 Classe SUMO . . . 233
3.28.16 Classe SURF . . . 234
3.28.17 Classe TCNL . . . 234

3.28.18 Classe TEXO . . . 235
3.29 Données relatives aux options de calcul . . . 236
3.29.1 Classe AMO . . . 236
3.29.2 Classe CFT . . . 237
3.29.3 Classe CMA . . . 238
3.29.4 Classe CR1 . . . 238
3.29.5 Classe CR2 . . . 238
3.29.6 Classe CRG . . . 238
3.29.7 Classe DPL . . . 239
3.29.8 Classe DTO . . . 239
3.29.9 Classe EFN . . . 239
3.29.10 Classe ENL . . . 239
3.29.11 Classe EST . . . 240
3.29.12 Classe FSC . . . 240
3.29.13 Classe FSR . . . 241
3.29.14 Classe HPE . . . 241
3.29.15 Classe INA . . . 242
3.29.16 Classe INI . . . 242
3.29.17 Classe INP . . . 244
3.29.18 Classe INT . . . 245
3.29.19 Classe INU . . . 245
3.29.20 Classe LIM . . . 246
3.29.21 Classe MUL . . . 247
3.29.22 Classe NDP . . . 247
3.29.23 Classe PTX . . . 248
3.29.24 Classe QAB . . . 248
3.29.25 Classe SRE . . . 249
3.29.26 Classe STK . . . 249
3.29.27 Classe STP . . . 250
3.29.28 Classe STT . . . 250
3.29.29 Classe STU . . . 250
3.29.30 Classe SUI . . . 251
3.29.31 Classe TXO . . . 251
3.30 Pré/post-traitement . . . 251
3.30.1 Classe MAIL . . . 252
3.30.2 Classe RSV4 . . . 254
3.31 Exemple de jeu de données . . . 254
3.31.1 Script . . . 254
3.31.2 Commentaires . . . 257

4 Installation **261**
4.1 Généralités . . . 261
4.2 Linux . . . 261
4.3 Windows . . . 262
4.4 Test de l'installation . . . 262

5 Bibliothèque d'exemples 263
5.1 Présentation générale . . . 263
5.2 Catégorie "structure statique linéaire linéique" (ssll) . . . 266
5.2.1 Test ssll001a . . . 266
5.2.2 Test ssll001b . . . 266
5.2.3 Test ssll002a . . . 267
5.2.4 Test ssll003a . . . 267
5.2.5 Test ssll003b . . . 268
5.2.6 Test ssll004a . . . 268
5.2.7 Test ssll005a . . . 269
5.2.8 Test ssll005b . . . 269
5.2.9 Test ssll006a . . . 270
5.2.10 Test ssll007 . . . 270
5.2.11 Test ssll008a . . . 271
5.2.12 Test ssll009a . . . 271
5.2.13 Test ssll010a . . . 272
5.2.14 Test ssll010b . . . 272
5.3 Catégorie "structure statique linéaire plane" (sslp) . . . 272
5.3.1 Test sslp001a . . . 272
5.3.2 Test sslp001b . . . 273
5.3.3 Test sslp001c . . . 273
5.3.4 Test sslp001d . . . 274
5.3.5 Test sslp001e . . . 274
5.3.6 Test sslp001f . . . 275
5.3.7 Test sslp001g . . . 275
5.3.8 Test sslp001h . . . 276
5.3.9 Test sslp001i . . . 276
5.3.10 Test sslp001j . . . 277
5.3.11 Test sslp001k . . . 277
5.3.12 Test sslp001l . . . 278
5.3.13 Test sslp001m . . . 279
5.3.14 Test sslp002a . . . 279
5.3.15 Test sslp002b . . . 280
5.3.16 Test sslp002c . . . 280
5.3.17 Test sslp002d . . . 281
5.3.18 Test sslp002e . . . 281
5.3.19 Test sslp002f . . . 282
5.3.20 Test sslp003 . . . 282
5.3.21 Test sslp004a . . . 282
5.3.22 Test sslp005a . . . 283
5.3.23 Test sslp005b . . . 283
5.3.24 Test sslp006a . . . 284
5.3.25 Test sslp006b . . . 284
5.3.26 Test sslp007a . . . 285
5.3.27 Test sslp008a . . . 285
5.3.28 Test sslp009a . . . 286
5.3.29 Test sslp010 . . . 286
5.3.30 Test sslp011a . . . 287
5.3.31 Test sslp011b . . . 287

5.3.32 Test sslp011c . . . 288
5.3.33 Test sslp012 . . . 288
5.3.34 Test sslp013 . . . 289
5.3.35 Test sslp014 . . . 289
5.3.36 Test sslp015 . . . 290
5.3.37 Test sslp016 . . . 290
5.4 Catégorie “structure statique linéaire surfacique” (ssls) . . . 291
5.4.1 Test ssls001 . . . 291
5.4.2 Test ssls002a . . . 291
5.4.3 Test ssls002b . . . 292
5.4.4 Test ssls002c . . . 292
5.4.5 Test ssls002d . . . 293
5.4.6 Test ssls003a . . . 293
5.4.7 Test ssls003b . . . 294
5.4.8 Test ssls003c . . . 294
5.4.9 Test ssls003d . . . 295
5.4.10 Test ssls004a . . . 295
5.4.11 Test ssls004b . . . 296
5.4.12 Test ssls005a . . . 296
5.4.13 Test ssls005b . . . 297
5.4.14 Test ssls006 . . . 297
5.4.15 Test ssls007a . . . 298
5.4.16 Test ssls008a . . . 298
5.4.17 Test ssls009a . . . 299
5.4.18 Test ssls009b . . . 299
5.4.19 Test ssls009c . . . 300
5.4.20 Test ssls010a . . . 300
5.4.21 Test ssls010b . . . 300
5.4.22 Test ssls011 . . . 301
5.5 Catégorie “structure statique linéaire volumique” (sslv) . . . 301
5.5.1 Test sslv001a . . . 301
5.5.2 Test sslv001b . . . 302
5.5.3 Test sslv001c . . . 302
5.5.4 Test sslv001d . . . 303
5.5.5 Test sslv001e . . . 303
5.5.6 Test sslv001f . . . 304
5.5.7 Test sslv001g . . . 304
5.5.8 Test sslv001h . . . 305
5.5.9 Test sslv002a . . . 305
5.5.10 Test sslv002b . . . 306
5.5.11 Test sslv003a . . . 306
5.5.12 Test sslv004a . . . 307
5.5.13 Test sslv005a . . . 307
5.5.14 Test sslv006a . . . 308
5.5.15 Test sslv007a . . . 308
5.5.16 Test sslv008a . . . 309
5.5.17 Test sslv008b . . . 309
5.5.18 Test sslv008c . . . 310
5.5.19 Test sslv008d . . . 310

5.6 Catégorie “structure statique non linéaire linéique” (ssnl) . . . 311
5.6.1 Test ssnl001a . . . 311
5.6.2 Test ssnl001b . . . 311
5.6.3 Test ssnl001c . . . 312
5.6.4 Test ssnl001d . . . 312
5.6.5 Test ssnl001e . . . 313
5.7 Catégorie “structure statique non linéaire plane” (ssnp) . . . 313
5.7.1 Test ssnp001a . . . 313
5.7.2 Test ssnp002a . . . 314
5.7.3 Test ssnp003a . . . 314
5.7.4 Test ssnp003b . . . 315
5.7.5 Test ssnp004a . . . 315
5.7.6 Test ssnp005a . . . 316
5.7.7 Test ssnp006a . . . 316
5.7.8 Test ssnp006b . . . 317
5.7.9 Test ssnp007 . . . 317
5.7.10 Test ssnp008a . . . 318
5.7.11 Test ssnp008b . . . 318
5.7.12 Test ssnp008c . . . 319
5.7.13 Test ssnp008d . . . 319
5.7.14 Test ssnp009a . . . 320
5.7.15 Test ssnp009b . . . 320
5.7.16 Test ssnp010a . . . 320
5.7.17 Test ssnp010b . . . 321
5.7.18 Test ssnp011a . . . 321
5.7.19 Test ssnp011b . . . 322
5.7.20 Test ssnp012 . . . 322
5.7.21 Test ssnp013 . . . 323
5.7.22 Test ssnp014a . . . 323
5.7.23 Test ssnp014b . . . 324
5.7.24 Test ssnp014c . . . 324
5.7.25 Test ssnp014d . . . 325
5.7.26 Test ssnp014e . . . 325
5.7.27 Test ssnp014f . . . 326
5.7.28 Test ssnp014g . . . 326
5.7.29 Test ssnp015a . . . 327
5.7.30 Test ssnp015b . . . 328
5.7.31 Test ssnp016a . . . 328
5.7.32 Test ssnp016b . . . 329
5.7.33 Test ssnp017 . . . 329
5.7.34 Test ssnp018 . . . 330
5.7.35 Test ssnp020 . . . 330
5.7.36 Test ssnp021 . . . 330
5.7.37 Test ssnp021b . . . 331
5.7.38 Test ssnp022 . . . 331
5.7.39 Test ssnp023 . . . 332
5.7.40 Test ssnp024a . . . 332
5.7.41 Test ssnp024b . . . 333
5.7.42 Test ssnp025 . . . 334

5.7.43 Test ssnp026 . . . 334
5.7.44 Test ssnp027a . . . 335
5.7.45 Test ssnp027b . . . 335
5.7.46 Test ssnp027c . . . 336
5.7.47 Test ssnp028 . . . 336
5.7.48 Test ssnp029 . . . 337
5.7.49 Test ssnp030 . . . 337
5.7.50 Test ssnp031 . . . 338
5.7.51 Test ssnp032 . . . 338
5.7.52 Test ssnp033 . . . 339
5.7.53 Test ssnp034 . . . 339
5.7.54 Test ssnp035 . . . 340
5.7.55 Test ssnp036 . . . 340
5.7.56 Test ssnp037 . . . 341
5.7.57 Test ssnp038 . . . 341
5.7.58 Test ssnp039 . . . 342
5.7.59 Test ssnp040 . . . 342
5.7.60 Test ssnp041 . . . 343
5.7.61 Test ssnp042 . . . 343
5.8 Catégorie “structure statique non linéaire surfacique” (ssns) . . . 343
5.8.1 Test ssns001a . . . 343
5.8.2 Test ssns001b . . . 344
5.8.3 Test ssns001c . . . 344
5.8.4 Test ssns001d . . . 345
5.8.5 Test ssns001e . . . 345
5.9 Catégorie “structure statique non linéaire volumique” (ssnv) . . . 346
5.9.1 Test ssnv001a . . . 346
5.9.2 Test ssnv002a . . . 346
5.9.3 Test ssnv002b . . . 347
5.9.4 Test ssnv003 . . . 347
5.9.5 Test ssnv004a . . . 348
5.9.6 Test ssnv004b . . . 348
5.9.7 Test ssnv004c . . . 349
5.9.8 Test ssnv004d . . . 349
5.9.9 Test ssnv005a . . . 349
5.9.10 Test ssnv005b . . . 350
5.9.11 Test ssnv005c . . . 350
5.9.12 Test ssnv005d . . . 351
5.9.13 Test ssnv006 . . . 351
5.9.14 Test ssnv007 . . . 352
5.9.15 Test ssnv008 . . . 353
5.10 Catégorie “structure dynamique linéaire linéique” (sdll) . . . 353
5.10.1 Test sdll001a . . . 353
5.10.2 Test sdll001b . . . 354
5.11 Catégorie “structure dynamique linéaire surfacique” (sdls) . . . 354
5.11.1 Test sdls001a . . . 354
5.11.2 Test sdls001b . . . 355
5.11.3 Test sdls001c . . . 355
5.11.4 Test sdls001d . . . 356

5.11.5 Test sdls002a . . . 356
5.11.6 Test sdls002b . . . 357
5.11.7 Test sdls002c . . . 357
5.11.8 Test sdls002d . . . 358
5.11.9 Test sdls002e . . . 358
5.11.10 Test sdls003a . . . 359
5.11.11 Test sdls003b . . . 359
5.11.12 Test sdls004a . . . 359
5.11.13 Test sdls004b . . . 360
5.12 Catégorie “structure dynamique linéaire volumique” (sdlv) . . . 360
5.12.1 Test sdlv001a . . . 360
5.12.2 Test sdlv002a . . . 361
5.13 Catégorie “structure transitoire non linéaire plane” (stnp) . . . 361
5.13.1 Test stnp001 . . . 361
5.13.2 Test stnp002 . . . 362
5.14 Catégorie “structure transitoire non linéaire volumique” (stnv) . . . 363
5.14.1 Test stnv001 . . . 363
5.14.2 Test stnv002 . . . 363
5.15 Catégorie “thermique permanent linéaire plane” (tplp) . . . 364
5.15.1 Test tplp001 . . . 364
5.16 Catégorie “thermique transitoire linéaire plane” (ttlp) . . . 364
5.16.1 Test ttlp001 . . . 364
5.17 Catégorie “thermique transitoire non linéaire plane” (ttnp) . . . 365
5.17.1 Test ttnp001 . . . 365
5.17.2 Test ttnp002 . . . 365
5.18 Catégorie “thermique transitoire non linéaire volumique” (ttnv) . . . 366
5.18.1 Test ttnv001 . . . 366
5.18.2 Test ttnv002 . . . 366
5.19 Catégorie “poreux transitoire linéaire plan” (ptlp) . . . 367
5.19.1 Test ptlp001 . . . 367
5.20 Catégorie “poreux transitoire non linéaire plan” (ptnp) . . . 367
5.20.1 Test ptnp001 . . . 367
5.21 Catégorie “poreux transitoire linéaire volumique” (ptlv) . . . 368
5.21.1 Test ptlv001 . . . 368
5.22 Catégorie “poreux transitoire non linéaire volumique” (ptnv) . . . 368
5.22.1 Test ptnv001 . . . 368
5.23 Catégorie “hydrogéologie permanente non linéaire plane” (hpnp) . . . 369
5.23.1 Test hpnp001 . . . 369
5.24 Catégorie “hydrogéologie transitoire non linéaire plane” (htnp) . . . 369
5.24.1 Test htnp001 . . . 369
5.24.2 Test htnp002 . . . 370
5.25 Catégorie “fonctionnalités diverses” (zzzz) . . . 370
5.25.1 Test zzzz001a . . . 370
5.25.2 Test zzzz001b . . . 371
5.25.3 Test zzzz001c . . . 371
5.25.4 Test zzzz002 . . . 372
5.25.5 Test zzzz003 . . . 372
5.25.6 Test zzzz004a . . . 372
5.25.7 Test zzzz004b . . . 373

5.25.8 Test zzzz004c 373
5.25.9 Test zzzz004d 373
5.25.10 Test zzzz005a 374
5.25.11 Test zzzz005b 374
5.25.12 Test zzzz006 374
5.25.13 Test zzzz007 375
5.25.14 Test zzzz008 375
5.25.15 Test zzzz009 376
5.25.16 Test zzzz010 376
5.25.17 Test zzzz011 376

Index des classes du langage Pilote **378**

Index des classes du langage CESAR **378**

Chapitre 1

Introduction

1.1 Contexte

Développé par l'Université Gustave Eiffel (ex-IFSTTAR, ex-LCPC)), le **solveur éléments finis CESAR** permet de pérenniser les modèles numériques issus des unités de recherche de l'Institut. Il fait également l'objet d'une commercialisation par la société Itech qui développe une **interface graphique (CLEO)** dédiée à son utilisation.

L'interface CLEO permet à l'utilisateur de saisir ses données graphiquement et de visualiser les résultats du calcul. Elle épargne donc à l'utilisateur la gestion des fichiers de communication avec le solveur CESAR :

- écriture du fichier texte (.data) contenant la mise en données du calcul,
- lecture du fichier binaire (.rsv4) contenant les résultats du calcul.

Des insuffisances. Néanmoins, une réflexion menée par le laboratoire LISIS (Laboratoire Instrumentation, Simulation et Informatique Scientifique) de l'IFSTTAR a identifié un certain nombre de besoins qui ne sont pas satisfaits par les fonctionnalités actuelles de l'ensemble CLEO/CESAR :

- utiliser CESAR sous Linux (CLEO ne fonctionne que sous Windows),
- pouvoir écrire et relire facilement le fichier de mise en données d'une étude CESAR (le format actuel du .data présente 2 inconvénients : beaucoup de données ne sont pas introduites par des mots-clé ce qui rend les blocs de données peu lisibles, et la plupart des propriétés du modèle doivent être fournies à l'échelle du noeud ce qui se révèle souvent très lourd voire irréalisable à la main),
- réaliser une étude paramétrique avec CESAR,
- utiliser des fonctionnalités de scripting pour définir une étude CESAR,
- réaliser une étude CESAR en utilisant un fichier de maillage existant, écrit dans un format autre que celui de CESAR,
- construire le maillage d'une étude CESAR à l'aide d'un ou plusieurs outils de maillage tiers,
- visualiser les résultats d'une étude CESAR à l'aide d'un ou plusieurs outils de visualisation tiers,
- réaliser un contre-calcul d'une étude CESAR avec un autre code de calcul,
- ...

L'émergence actuelle de nombreux outils scientifiques (mailleurs, viewers, codes de calculs ...) du monde du logiciel libre rend ces insuffisances encore plus pregnantes.

Une première solution. En conséquence, le groupe des Modèles Numériques a ouvert un nouvel axe de travail autour du solveur CESAR consistant à développer des solutions sur les problématiques de maillage et de mise en données (**en amont du solveur**) et d'analyse intéractive des résultats générés (**en aval du solveur**).
Un premier choix a consisté à développer un unique outil, le **Pilote de CESAR**, indépendant du solveur et destiné à apporter les fonctionnalités suivantes :

- un **langage de commandes** simplifiant la mise en données textuelle d'une étude (lisibilité, paramétrage et scripting),
- l'**échange de données** avec des outils tiers (mailleur, viewer, code éléments finis).

Comme l'interface CLEO, le Pilote gère en interne les fichiers de données et de résultats au format natif du solveur (fichiers .data et .rsv4).

1.2 Vue d'ensemble

Deux langages spécifiques. Le Pilote de CESAR permet de définir une étude CESAR en écrivant un script en langage **Python**. De façon précise, la mise en données se fait en utilisant un **langage spécifique** au Pilote que l'on peut voir comme une extension de Python ou comme un "module" au sens de Python, tout en utilisant en tant que de besoin (pour faire du scripting notamment) l'ensemble des fonctionnalités de base du langage Python.
Le pilote propose 2 langages spécifiques correspondant à 2 niveaux d'abstraction différents :

- le premier niveau, appelé **langage de commandes CESAR**, est très proche de l'interface actuelle du solveur décrite dans son "Manuel de référence". Il permet de décrire finement une étude CESAR en utilisant exactement les mêmes informations que celles fournies dans le fichier .data actuel mais présentées sous une forme un peu différente. Ce langage possède donc le même bas niveau d'abstraction que l'interface actuelle de CESAR. La syntaxe de ce langage est décrite à la section 3.
- le deuxième niveau, appelé **langage de commandes Pilote** présente un niveau d'abstraction plus élevé. En particulier, il permet d'affecter les propriétés du modèle à des groupes d'entités du maillage (sous réserve que ces groupes soient présents dans la définition du maillage). Dans la mesure du possible, il est indépendant du solveur CESAR et de fait il est possible de l'utiliser (dans la limite des fonctionnalités disponibles) pour réaliser un calcul avec le solveur éléments finis **Code_Aster Libre**. Le maillage peut être construit avec un mailleur tiers et les résultats visualisés avec un viewer tiers. Dans sa version actuelle, le langage reconnait le format du logiciel libre **GMSH** à la fois en pré et en post traitement. Il reconnait également le format **MED** utilisé notamment par **Code_Aster Libre** et par la plate-forme libre de pré-post traitement **SALOME**. Il est également possible de définir le maillage "à la main" dans le script Python. La syntaxe de ce langage est décrite à la section 2.

Chacun de ces 2 langages présente un intérêt intrinsèque :

- le langage Pilote permet une mise en données présentant un niveau d'abstraction suffisamment élevé pour envisager l'export de ces données vers différents solveurs éléments finis et pour permettre l'interfaçage avec des outils de maillage et de visualisation,
- le langage CESAR permet de piloter finement le solveur et peut être vu comme une évolution de l'interface actuelle facilitant la mise en donnée en la rendant plus lisible et en étendant ses possibilités en permettant le scripting.

A noter qu'en interne au Pilote, du point de vue informatique, le langage CESAR est systématiquement utilisé par le Pilote comme API pour la génération du fichier de données (.data) de CESAR.

Modèle objet. Du point de vue technique, ces 2 langages constituent 2 modèles de données orientés objet. Les éléments du langage sont définis par des classes, des attributs et des méthodes au sens de la programmation orientée objet (POO). Néanmoins, pour utiliser le langage, il suffit de consulter la syntaxe définie aux sections 2 et 3 sans que cela nécessite des connaissances approfondies en POO.
Donnons cependant quelques définitions de base qui facilitent la lecture de la syntaxe du langage.

- une **classe** peut être vue comme une façon de regrouper un ensemble de données, éventuellement de natures différentes (type "composite").
- les données élémentaires regroupées dans une classe sont appelées **attributs** et peuvent présenter un type simple ou bien un type composite.
- lorsque l'utilisateur utilise une classe du langage pour renseigner et stocker des données, on dit qu'il **instancie** et renseigne la classe. Le résultat de cette opération est un **objet** que l'utilisateur doit nommer de façon à pouvoir l'utiliser plus loin dans son script.
- une classe comporte des données mais également des **méthodes**, c'est à dire des fonctionnalités, que l'utilisateur peut appeler à partir d'un objet de la classe. Par exemple, la classe **ETUDE** du langage Pilote dispose de la méthode **lancer** qui permet de lancer le calcul CESAR.
- certaines classes sont reliées entre elles par un lien dit d'héritage. On dit que la classe B **hérite** de la classe A pour signifier que la classe B est une sorte de classe A. On dit aussi que la classe A est la **classe de base** de la classe B ou encore que la classe B **dérive** de la classe A.
- la syntaxe du langage fait parfois apparaître, par commodité, des classes dites **abstraites**. L'utilisateur ne peut pas instancier des objets à partir de ces classes. Il ne peut qu'instancier des classes qui héritent (dérivent) de ces classes abstraites. Par exemple, la classe **COMPORTEMENT** du langage Pilote est abstraite, et l'utilisateur ne peut qu'instancier des classes dérivées correspondant à des lois de comportement particulières comme la classe **ELAS_LINE_ISO** pour l'élasticité linéaire isotrope. La syntaxe du langage précise systématiquement le caractère abstrait ou instanciable des classes.

Syntaxe des commandes et conventions typographiques. Pour chaque classe, on donne la syntaxe de son **constructeur** c'est à dire les règles à respecter pour instancier la classe. On précise en particulier le caractère obligatoire ou facultatif des mots-clé ainsi que les types des valeurs qui peuvent être attribuées à ces mots-clé.
La plupart des attributs d'une classe sont renseignés via le constructeur, donc par l'utilisateur, mais certaines classes possèdent des attributs qui sont renseignés en interne par le Pilote. Par exemple, la classe **MAILLAGE** gère en interne l'attribut **mesh** qui stocke le maillage, ou bien encore la classe **RESULTAT** qui gère en interne l'attribut **fields** qui stocke les champs de résultats.
La syntaxe d'un constructeur est décrite en utilisant les conventions typographiques suivantes (empruntées à la documentation du langage de commandes du solveur Code_Aster Libre) pour le renseignement des mots-clé :

♦ : pour signifier qu'un mot-clé est obligatoire,
◊ : pour signifier qu'un mot-clé est facultatif,
/ : pour signifier qu'un mot-clé doit prendre une seule valeur parmi la liste des valeurs proposées,
| : pour signifier qu'un mot-clé peut prendre une ou plusieurs valeurs parmi la liste des valeurs proposées,
[...] : pour préciser le type de valeur que peut prendre un mot clé, par exemple :

- pour un nombre réel : [**float**]
- pour une instance de la classe **MAILLAGE** : [**MAILLAGE**]

On utilise également un code couleur :

- **rouge** pour le nom d'une classe ou pour un type élémentaire, par exemple : **FICHIER** ou **float**,
- **bleu** pour le nom des attributs d'une classe présents dans son constructeur, par exemple : **TYP** ou **DIM**,
- **vert** pour les valeurs des littéraux d'énumération que peuvent prendre certains attributs, par exemple : **'GMSH'** ou **3**.

Objectif et organisation du présent document. Le présent document constitue la documentation utilisateur du Pilote. Il a pour objectif d'en présenter :

- l'ensemble des fonctionnalités disponibles ainsi que la syntaxe de l'ensemble des commandes :
  - du langage Pilote (chapitre 2),
  - du langage CESAR (chapitre 3).

  Les chapitres 2 et 3 se terminent chacun par un exemple complet de script assorti de commentaires détaillant les différentes parties de la mise en données.
- les instructions d'installation (chapitre 4)
- l'ensemble des tests de non régression créés à l'occasion du développement du Pilote de CESAR et qui peuvent tenir lieu d'exemples illustrant l'emploi des 2 langages spécifiques proposés. De façon à guider l'utilisateur dans la recherche d'exemples d'utilisation de fonctionnalités données, le chapitre 5 propose ainsi une vue synoptique de l'ensemble des tests. Ces tests ne constituent cependant pas de véritables études au sens où certaines valeurs numériques (dimensions géométriques, propriétés des matériaux, charges, ...) ne correspondent pas toujours à des situations réalistes.

Ce document ne se substitue pas à la documentation du solveur CESAR (manuels d'utilisation, de référence, ...) à laquelle il convient donc de se référer en cas de besoin.

Chapitre 2

Langage de commandes Pilote

Le langage Pilote permet :

- de piloter le solveur CESAR, c'est à dire de générer son fichier de données (.data), de lancer son exécution et de lire son fichier de résultats (.rsv4), au moyen d'un script Python utilisant les commandes du langage.
- d'interfacer le solveur CESAR avec des outils de maillage et de visualisation de résultats. Actuellement, l'interfaçage avec le logiciel libre GMSH et avec le format MED (donc avec la plate-forme libre SALOME) est disponible.
- d'exporter les données d'une étude vers un autre solveur éléments finis que CESAR. Actuellement, seul l'export (partiel) vers le solveur Code_Aster Libre (version 11.1) est disponible.

Les commandes du langage ne sont pas "adhérentes" à CESAR et présentent un niveau d'abstraction plus élevé que celui du fichier de données CESAR (.data). Ce niveau est comparable à celui du fichier de commandes ASTER (.comm).

Le langage Pilote peut être utilisé de façon directe (interface textuelle) ou indirecte (interface graphique).

Interface textuelle. La mise en données d'une étude s'effectue en écrivant un script Python (par exemple `etude.py`). Du point de vue Python, le langage Pilote est vu comme un "module" que l'on doit importer en tête du script Python en écrivant la ligne :

```
from modele_donnees import *
```

Lorsqu'on a besoin d'utiliser l'API Python de MED Mémoire (par exemple pour définir un maillage directement sans passer par un mailleur), incluse dans le langage Pilote, il est nécessaire de l'importer également en ajoutant la ligne :

```
from libMEDMEM_Swig import *
```

L'exécution de l'étude se fait en appelant (dans un terminal) le script `run_pilote` (présent dans le répertoire `bin/` de la distribution du Pilote) et en lui passant le script des données :
`> run_pilote etude.py`

Interface graphique. La mise en donnée de l'étude peut également s'effectuer de façon graphique en appelant le script `ihm_pilote` (présent dans le répertoire `bin/` de la distribution du Pilote) qui lance un outil intéractif permettant de saisir et de visualiser graphiquement (sous forme d'arbre) l'ensemble des données d'une étude, ainsi que de générer automatiquement le script Python correspondant aux données saisies.

Exemple A titre d'exemple, voici comment lancer le cas test ssnp001a présent dans le répertoire `pilote/tests/donnees`. Ci-dessous, `cespil` désigne le répertoire d'installation du Pilote et `home` désigne le répertoire de l'utilisateur.
De façon préalable :

- on crée un répertoire de travail, par exemple du nom du test : `home/ssnp001a`
- on y copie les fichiers en entrée du test : fichier de maillage `ssnp001a.msh` et script `ssnp001a.py`
- on édite le script et on met à jour la variable `rep_trav` : `rep_trav = 'home/ssnp001a/'`

Pour lancer l'exécution, on utilise le script `run_pilote` :

- on ouvre un terminal et on se place dans le répertoire de travail : `cd home/ssnp001a`
- on appelle le script : `cespil/bin/run_pilote ssnp001a.py`

De façon alternative, on peut utiliser le script `ihm_pilote` :

- on lance l'IHM : `cespil/bin/ihm_pilote`
- on lance l'exécution simplement en important le script avec l'explorateur : `File` / `Import` / `ssnp001a.py`

2.1 Principe général de mise en données

L'idée principale consiste à instancier et renseigner un modèle de données à partir des commandes du langage.
On doit ainsi définir une **ETUDE** qu'on peut ensuite **lancer**, c'est à dire exécuter à l'aide d'un solveur éléments finis. Une étude peut comporter plusieurs appels au solveur (CESAR, ...), c'est à dire plusieurs **RESOLUTION**. Ces résolutions peuvent être indépendantes et dans ce cas il est loisible de les distribuer sur plusieurs processeurs. Par exemple, il est ainsi possible de réaliser une étude paramétrée.
Une **RESOLUTION** doit contenir l'ensemble des données nécessaires pour pouvoir appeler un solveur éléments finis, c'est dire classiquement les données relatives :

- au maillage,
- au modèle (type d'élément fini),
- aux matériaux (types de lois de comportement),
- aux caractéristiques (géométries des éléments de structures, ...),
- aux liaisons (contact, relations linéaires entre degrés de liberté, ...)
- aux conditions limites,
- aux cas de chargements,
- à l'analyse (statique linéaire ou non linéaire, ...)

En pratique, on commence donc par définir des ensembles de données de ce type à l'aide des classes :

- **MAILLAGE**,
- **MODELE**,
- **CARACTERISTIQUE**,
- **MATERIAU**,
- **LIAISON**,
- **COND_LIMITE**,
- **CAS_CHARGEMENT** (c'est à dire une combinaison linéaire de **CHARGEMENT**),
- **ANALYSE**,

ce qui consiste en général à affecter des propriétés à des lieux du maillage ou du modèle à l'aide de la classe générique **AFFECT**.
Puis on peut alors construire une ou plusieurs **RESOLUTION** en invoquant des références aux objets précédents construits.

Ensuite, on définit les données de résultats à l'aide de la classe **RESULTAT**, ce qui consiste soit à extraire des grandeurs sur des lieux du maillage (champs de grandeurs) à l'aide de la classe générique **EXTRACT**, soit à définir un fichier d'export de l'ensemble des résultats pour visualisation avec un outil externe.
Enfin, on construit une **ETUDE** faisant référence aux maillages, résolutions et résultats définis précédemment. C'est ainsi qu'une étude peut contenir plusieurs résolutions (ou cas d'étude) qui peuvent éventuellement être distribuées sur plusieurs processeurs à l'aide de la classe **PARAM_EXEC** permettant de définir les paramètres d'exécution.
On lance l'exécution de l'étude à l'aide la méthode **lancer** de la classe **ETUDE**.

2.2 Lecture et écriture sur fichier

CESAR est un solveur éléments finis et ne possède donc pas de fonctionnalités propres de mailleur ou de viewer de résultats. Un des objectifs du Pilote consiste précisément à interfacer CESAR avec des logiciels externes dotés de telles fonctionnalités.
La démarche standard consiste donc à :

- générer un fichier de maillage à l'aide d'un mailleur externe puis importer ce fichier depuis le Pilote,
- piloter le solveur CESAR (générer son fichier de données, lancer son éxécution et lire son fichier de résultats),
- visualiser les résultats du calcul CESAR à l'aide d'un viewer externe sur la base d'un fichier de résultats exporté par le Pilote au format adapté.

La déclaration de ces fichiers se fait à l'aide de la classe **FICHIER**.

2.2.1 Classe FICHIER

La classe **FICHIER** est la classe permettant de définir un fichier par son nom et par son format.

```
fic = FICHIER(
    ♦ NOM = nom, [str]
    ♦ FORMAT = / 'GMSH',
               / 'MED',
               / 'ASTER',
               / 'CESAR',
               / 'TXT',
    ) # fin FICHIER
```

NOM : chaîne de caractères définissant le nom complet du fichier.
FORMAT : chaîne de caractères définissant le format du fichier, à choisir parmi :

- **'GMSH'** : pour un fichier de maillage ou un fichier de résultats au format du logiciel GMSH (`http://www.geuz.org/gmsh/`),
- **'MED'** : pour un fichier de maillage ou un fichier de résultats au format MED, (`http://www.salome-platform.org/`)
- **'ASTER'** : pour un fichier de maillage ou un fichier de données au format du solveur éléments finis Code_Aster d'EDF (`http://www.code-aster.org`),
- **'CESAR'** : pour un fichier de résultats ou un fichier de données au format du solveur éléments finis CESAR,
- **'TXT'** : pour un fichier de résultats au format texte.

Suivant le contexte, seules certaines possibilités parmi celles listées pour **FORMAT** sont autorisées.

Exemples

```
from modele_donnees import *

fic_in = FICHIER(NOM = 'tests/atelier/ssnp001a.msh'
                 FORMAT = 'GMSH')
```

On peut exploiter l'environnement Python et utiliser des variables :

```
from modele_donnees import *

rep_trav = 'tests/atelier/'
fic_msh = rep_trav + 'ssnp001a.msh'

fic_in = FICHIER(NOM = fic_msh,
                 FORMAT = 'GMSH')
```

2.3 Affectation de propriétés

Une large part de la mise en données consiste à appliquer des propriétés sur des lieux du maillage ou du modèle. Ces propriétés peuvent être des formulations d'éléments finis, des caractéristiques de matériaux, des caractéristiques géométriques, des conditions limites, des chargements.
L'affectation de propriétés se fait de façon générique à l'aide de la classe **AFFECT** qui possède un attribut de type **PROPRIETE** et un attribut de type **LIEU**.

2.3.1 Classe AFFECT

La classe **AFFECT** est la classe générique permettant d'affecter une propriété à un lieu.

affect = **AFFECT**(
- ♦ **PROPRIETE** = ppte, [**PROPRIETE**]
- ♦ **LIEU** = lieu, [**LIEU**]

) # *fin AFFECT*

PROPRIETE : la propriété à affecter.
LIEU : le lieu d'affectation de la propriété.

2.3.2 Classe PROPRIETE

La classe **PROPRIETE** est une classe abstraite. Elle constitue la classe de base des classes suivantes :

- **FORMULATION** décrite à la section relative aux modèles 2.9,
- **COMPORTEMENT** décrite à la section relative aux matériaux 2.10,
- **GEOMETRIE** décrite à la section relative aux caractéristiques 2.11,
- **RELA_LINE** décrite à la section relative aux liaisons 2.13,
- **MATR_ELEM** décrite à la section relative aux liaisons 2.13,
- **CONTACT** décrite à la section relative aux liaisons 2.13,
- **JOINT** décrite à la section relative aux liaisons 2.13,
- **CONDITION** décrite à la section relative aux conditions limites 2.14,
- **CHARGE** décrite à la section relative aux chargements 2.15.

2.3.3 Classe LIEU

La classe **LIEU** permet de définir une entité ou un groupe d'entités du maillage (noeuds, mailles, faces, arêtes) ou du modèle (fibres d'une poutre multi-fibre ou couches d'une coque multi-couches).

lieu = **LIEU**(
- **NOMS** = noms, [**list<str>**]
- **TYP** = / **'GROUP_MAILLE'**,
  / **'GROUP_FACE'**,
  / **'GROUP_ARETE'**,
  / **'GROUP_NOEUD'**,
  / **'MAILLE'**,
  / **'FACE'**,
  / **'ARETE'**,
  / **'NOEUD'**,
  / **'GROUP_FIBRE'**,
  / **'GROUP_COUCHE'**,

) # *fin LIEU*

NOMS : liste de chaînes de caractères définissant des noms d'entités de lieu.
TYP : chaîne de caractères définissant le type des entités de lieu, à choisir parmi :
— **'GROUP_MAILLE'** : pour des groupes de mailles du maillage,
— **'GROUP_FACE'** : pour des groupes de faces du maillage,
— **'GROUP_ARETE'** : pour des groupes d'arêtes du maillage,
— **'GROUP_NOEUD'** : pour des groupes de noeuds du maillage,
— **'MAILLE'** : pour des mailles du maillage,
— **'FACE'** : pour des faces du maillage,
— **'ARETE'** : pour des arêtes du maillage,
— **'NOEUD'** : pour des noeuds du maillage,
— **'GROUP_FIBRE'** : pour des groupes de fibres (cas d'un calcul avec des poutres multi-fibres),
— **'GROUP_COUCHE'** : pour des groupes de couches (cas d'un calcul avec des coques multi-couches).
Suivant le contexte, seules certaines possibilités parmi celles listées pour **TYP** sont autorisées.

Exemples

```
from modele_donnees import *

lieu = LIEU(NOMS = ['Surf1', 'Surf2'],
            TYP = 'GROUP_MAILLE')
```

Si la liste **NOMS** ne comporte qu'un seul élément, on peut omettre les crochets :

```
from modele_donnees import *

lieu = LIEU(NOMS = 'Surf',
            TYP = 'GROUP_MAILLE')
```

2.4 Extraction de grandeurs

Une partie de la mise en données consiste à définir la nature et la localisation dans le maillage des grandeurs que l'on souhaite extraire de la base de résultats.
L'extraction de grandeurs se fait de façon générique à l'aide de la classe **EXTRACT** qui possède un attribut de type **GRANDEUR** et un attribut de type **LIEU**.
On peut également extraire des grandeurs d'un **MAILLAGE** (coordonnées) ou d'un **RESULTAT** (déplacements, contraintes, ...) et les récupérer directement dans le script sous forme de dictionnaires Python, de façon à réaliser des calculs de post-traitement ou des impressions, ...
Ceci se fait via les méthodes **extraire()** des classes **MAILLAGE** et **RESULTAT** décrites dans les sections relatives aux maillages 2.8 et aux résultats 2.19.

2.4.1 Classe EXTRACT

La classe **EXTRACT** est la classe générique permettant d'extraire une grandeur sur un lieu.

extract = **EXTRACT**(
♦ **GRANDEUR** = grandeur, [**GRANDEUR**]
♦ **LIEU** = lieu, [**LIEU**]
) # *fin EXTRACT*

GRANDEUR : la grandeur à extraire.
LIEU : le lieu d'extraction de la grandeur.

2.4.2 Classe GRANDEUR

La classe **GRANDEUR** est une classe abstraite. C'est la classe de base des classes instanciables suivantes :

- **COOR_2D**,
- **COOR_3D**,
- **DEPLA_2D**,
- **DEPLA_3D**,
- **DEPLA_2D_ROTA**,
- **DEPLA_3D_ROTA**,
- **ACCEL_2D**,
- **ACCEL_3D**,
- **CONTR_2D**,
- **CONTR_3D**,
- **REAC_2D**.

Ces classes sont décrites dans les sections relatives aux maillages 2.8 et aux résultats 2.19.

2.5 Définition d'une étude

2.5.1 Classe ETUDE

La classe **ETUDE** est la classe permettant de définir l'étude à exécuter.

etude = **ETUDE**(
♦ **MAIL** = mail, [**list<MAILLAGE>**]

♦ **RESOL** = resol, [**list<RESOLUTION>**]
♦ **EXEC** = exec, [**PARAM_EXEC**]
◊ **RESU** = resu, [**list<RESULTAT>**]
) # *fin ETUDE*

MAIL : liste d'objets de type **MAILLAGE** définissant l'ensemble des maillages associés à l'étude.
RESOL : liste d'objets de type **RESOLUTION** définissant l'ensemble des résolutions associées à l'étude.
EXEC : objet de type **PARAM_EXEC** définissant les paramètres d'exécution de l'étude.
RESU : liste d'objets de type **RESULTAT** définissant l'ensemble des résultats associés à l'étude.

Paramètres d'exécution. La classe **PARAM_EXEC** est la classe permettant de définir les paramètres d'exécution de l'étude.

param = **PARAM_EXEC**(
♦ **TYP** = / **'DISTRIB'**,
♦ **NB_PROC** = nb_proc [**int**]
/ **'SEQUENT'**,
) # *fin PARAM_EXEC*

TYP : chaîne de caractères définissant le type d'exécution à choisir parmi :
— **'DISTRIB'** : pour une exécution distribuée sur plusieurs processeurs,
— **'SEQUENT'** : pour une exécution séquentielle sur un seul processeur.
NB_PROC : entier donnant le nombre de processeurs.

Méthodes. La classe **ETUDE** est munie des méthodes suivantes :
lancer() : méthode permettant de lancer l'exécution de l'étude par appel au solveur CESAR.
On peut lui passer de façon facultative 2 paramètres contrôlant les ressources utilisées par CESAR :
— nb_max_mots_reel : entier (50 000 000 par défaut) donnant la taille maximale du tableau de réels à l'intérieur duquel CESAR stocke l'information,
— nb_max_elem : entier (30 000 par défaut) donnant le nombre maximal d'éléments du modèle.
lancer_aster() : méthode permettant de lancer l'exécution de l'étude par appel au solveur ASTER.
On lui passe en paramètre :
— repertoire : chaîne de caractères donnant le nom du répertoire où on veut récupérer les sorties ASTER.
On peut également lui passer de façon facultative 2 paramètres contrôlant les ressources utilisées par ASTER :
— mem_jeveux : réel (8.0 par défaut) donnant la mémoire totale utilisée pour le calcul (= 8.mem_jeveux Mo donc 64 Mo par défaut),
— tp_max : entier (60 par défaut) donnant le temps maximal du calcul (en secondes).

Se reporter à la documentation U1.04.00 (Interface d'accès à Code_Aster : astk) d'ASTER pour plus de précision.
Contrairement à la méthode **lancer()**, les sorties d'ASTER ne sont pas relues par le Pilote.
exporter_donnees() : méthode permettant de générer les fichiers de données pour un solveur donné.
On lui passe en paramètres :
— code_calcul : chaîne de caractères donnant le nom du solveur à choisir parmi **'CESAR'** ou **'ASTER'**,
— repertoire : chaîne de caractères donnant le nom du répertoire où on veut récupérer les fichiers de données.

2.5.2 Classe PHASAGE

Une des particularités du solveur CESAR est de permettre la modélisation du phasage de construction des structures géotechniques, notamment pour modéliser l'interaction sol-structure et pour prédire la stabilité des sols et des fondations. En pratique, il convient d'écrire autant de jeux de données (fichiers .data) que de phases de construction puis de les exécuter successivement.
Ce mode de fonctionnement rentre donc naturellement dans la logique du langage Pilote où une **ETUDE** peut contenir plusieurs **RESOLUTION** pouvant être exécutées de façon séquentielle.
Néanmoins, pour simplifier la mise en données et garantir la cohérence des données utilisateur, 2 classes spécifiques ont été créées, **PHASAGE** et **PHASE**, dérivant respectivement des classes **ETUDE** et **RESOLUTION**. Ces 2 classes possèdent des constructeurs très comparables à ceux de leurs parents.

phasage = **PHASAGE**(
♦ **MAIL** = mail, [list<**MAILLAGE**>]
♦ **PHASE** = resol, [list<**PHASE**>]
◊ **RESU** = resu, [list<**RESULTAT**>]
) # *fin PHASAGE*

MAIL : liste d'objets de type **MAILLAGE** définissant l'ensemble des maillages associés au phasage.
PHASE : liste d'objets de type **PHASE** définissant l'ensemble des phases constituant le phasage.
RESU : liste d'objets de type **RESULTAT** définissant l'ensemble des résultats associés au phasage.
Ainsi, un **PHASAGE** ne possède pas la données **EXEC** dans son constructeur, car il est implicite que l'exécution doit ici être réalisée de façon séquentielle et non pas distribuée.

2.6 Génération d'un maillage à partir d'un domaine

2.6.1 Classe DOMAINE

La classe **DOMAINE** est la classe permettant de :
— définir la géométrie du domaine d'étude,
— définir des directives de maillage de ce domaine,

— générer le fichier de maillage associé.

Le fichier généré peut ensuite être lu par la classe **MAILLAGE**.

```
dom = DOMAINE(
        ♦ NOM = nom, [str]
        ♦ DIM = / 1,
                / 2,
                / 3,
        ♦ FIC_OUT = FICHIER(
                        ♦ FORMAT = / 'GMSH',
                        ♦ NOM = nom, [str]
                        ), # fin FICHIER
        ♦ OPER = oper, [list<OPERATION>]
        ◊ VISU = / 'OUI',
                 / 'NON',
        ) # fin DOMAINE
```

NOM : chaîne de caractères définissant le nom du domaine.
DIM : entier définissant la dimension du maillage : **1** (resp. **2**, **3**) pour un maillage linéique (resp. surfacique, volumique).
FIC_OUT : objet de type **FICHIER** définissant le fichier de domaine à exporter. Actuellement, seul le format **'GMSH'** est disponible. Le fichier de domaine correspond au fichier de géométrie pour GMSH (.geo).
OPER : liste d'objets de type **OPERATION** donnant les différentes instructions pour construire puis mailler le domaine.
VISU : chaîne de caractères permettant de visualiser (**'OUI'**) ou pas (**'NON'**) le domaine. La visualisation se fait via l'interface graphique de GMSH.

Méthodes

mailler() : méthode permettant de générer le maillage en spécifiant, via un argument **FIC_OUT** ([**FICHIER**]), le nom et le format du fichier de maillage. Actuellement, seul le format **'GMSH'** est disponible.

2.6.2 Classe OPERATION

Une opération permet des créer des entités géométriques (points, lignes, surfaces, volumes, ...), de les manipuler (extrusions, transformations, ...) et de leur affecter des directives de maillage (structure et finesse de parties du maillage, ...). Elle permet également de constituer des blocs d'entités qui pourront ensuite être invoqués pour y affecter les propriétés du modèle (propriétés physiques, conditions limites, chargements, ...).
La classe **OPERATION** est une classe abstraite. Elle constitue la classe de base des classes abstraites suivantes :

— **POINT**,
— **LIGNE**,
— **CERCLE**,
— **ELLIPSE**,
— **SPLINE**,
— **BSPLINE**,

- **SURFACE**,
- **VOLUME**,
- **BOUCLE**,
- **ASSEMBLAGE**,
- **BLOC**,
- **EXTRUSION**,
- **TRANSFORMATION**,
- **STRUCTURE**,
- **RECOMBINE**,
- **COHERENCE**,
- **FINESSE**,
- **DESTRUCTION**,
- **OPTION**,
- **ENTITES**,
- **ID**.

décrites à la section relative aux domaines 2.7,

2.7 Données relatives aux domaines

2.7.1 Classe POINT

La classe **POINT** est la classe permettant de définir un point.

poi = **POINT**(
- ♦ **ID** = id, [**int**] ou [**str**]
- ♦ **X** = x, [**float**]
- ♦ **Y** = y, [**float**]
- ♦ **Z** = z, [**float**]
- ◊ **H** = h, [**float**]

) # *fin POINT*

ID : entier ou chaîne de caractères identifiant le point.
X : réel donnant la coordonnée du point selon l'axe Ox.
Y : réel donnant la coordonnée du point selon l'axe Oy.
Z : réel donnant la coordonnée du point selon l'axe Oz.
H : réel donnant la finesse du maillage au voisinage du point.

2.7.2 Classe LIGNE

La classe **LIGNE** est la classe permettant de définir une ligne droite.

lign = **LIGNE**(
- ♦ **ID** = id, [**int**] ou [**str**]
- ♦ **P1** = p1, [**int**] ou [**str**]
- ♦ **P2** = p2, [**int**] ou [**str**]

) # *fin LIGNE*

ID : entier ou chaîne de caractères identifiant la ligne droite.

P1 : entier ou chaîne de caractères référençant le point de début de la ligne droite.
P2 : entier ou chaîne de caractères référençant le point de fin de la ligne droite.

2.7.3 Classe CERCLE

La classe **CERCLE** est la classe permettant de définir un arc de cercle d'angle inférieur à π.

cerc = **CERCLE**(
- **ID** = id, [**int**] ou [**str**]
- **P1** = p1, [**int**] ou [**str**]
- **PC** = pc, [**int**] ou [**str**]
- **P2** = p2, [**int**] ou [**str**]

) # *fin CERCLE*

ID : entier ou chaîne de caractères identifiant l'arc de cercle.
P1 : entier ou chaîne de caractères référençant le point de début de l'arc de cercle.
PC : entier ou chaîne de caractères référençant le point centre du cercle.
P2 : entier ou chaîne de caractères référençant le point de fin de l'arc de cercle.

2.7.4 Classe ELLIPSE

La classe **ELLIPSE** est la classe permettant de définir un arc d'ellipse.

elli = **ELLIPSE**(
- **ID** = id, [**int**] ou [**str**]
- **P1** = p1, [**int**] ou [**str**]
- **PC** = pc, [**int**] ou [**str**]
- **PGA** = pga, [**int**] ou [**str**]
- **P2** = p2, [**int**] ou [**str**]

) # *fin ELLIPSE*

ID : entier ou chaîne de caractères identifiant l'arc d'ellipse.
P1 : entier ou chaîne de caractères référençant le point de début de l'arc d'ellipse.
PC : entier ou chaîne de caractères référençant le point centre de l'ellipse.
PGA : entier ou chaîne de caractères référençant un point sur le grand axe de l'ellipse.
P2 : entier ou chaîne de caractères référençant le point de fin de l'arc d'ellipse.

2.7.5 Classe SPLINE

La classe **SPLINE** est la classe permettant de définir une courbe spline.

spli = **SPLINE**(
- **ID** = id, [**int**] ou [**str**]
- **ENT** = ent, [**list<int ou str>**]

) # *fin SPLINE*

ID : entier ou chaîne de caractères identifiant la courbe.
ENT : liste d'entiers ou chaînes de caractères référençant les points de contrôle de la courbe.

2.7.6 Classe BSPLINE

La classe **BSPLINE** est la classe permettant de définir une courbe B-spline.

bspli = **BSPLINE**(
♦ **ID** = id, [**int**] ou [**str**]
♦ **ENT** = ent, [**list<int ou str>**]
) # *fin BSPLINE*

ID : entier ou chaîne de caractères identifiant la courbe.
ENT : liste d'entiers ou chaînes de caractères référençant les points de contrôle de la courbe.

2.7.7 Classe BOUCLE

La classe **BOUCLE** est la classe permettant de définir une boucle orientée d'entités de même type.

bcle = **BOUCLE**(
♦ **ID** = id, [**int**]
♦ **TYP_ENT** = / **'LIGNE'**,
/ **'SURFACE'**,
♦ **ENT** = ent, [**list<int ou str>**]
) # *fin BOUCLE*

ID : entier identifiant la boucle.
TYP_ENT : chaîne de caractères spécifiant le type des entités constitutives de la boucle parmi **'LIGNE'** et **'SURFACE'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités constitutives de la boucle.

2.7.8 Classe SURFACE

La classe **SURFACE** est la classe permettant de définir une surface.

surf = **SURFACE**(
♦ **ID** = id, [**int**] ou [**str**]
♦ **BOUCLES** = ent, [**list<int>**]
◊ **REGLEE** = / **'OUI'**,
/ **'NON'**,
) # *fin SURFACE*

ID : entier ou chaîne de caractères identifiant la surface.
BOUCLES : liste d'entiers référençant les boucles de lignes bornant la surface.
REGLEE : chaîne de caractères spécifiant le caractère réglé (**'OUI'**) ou pas (**'NON'**) de la surface. Dans ce dernier cas la surface est plane.

2.7.9 Classe VOLUME

La classe **VOLUME** est la classe permettant de définir un volume.

volu = **VOLUME**(
♦ **ID** = id, [**int**] ou [**str**]
♦ **BOUCLES** = ent, [**list<int>**]
) # *fin VOLUME*

ID : entier ou chaîne de caractères identifiant le volume.
BOUCLES : liste d'entiers référençant les boucles de surface bornant le volume.

2.7.10 Classe ASSEMBLAGE

La classe **ASSEMBLAGE** est la classe permettant de définir une entité d'un certain type par assemblage d'entités de ce type.

asse = **ASSEMBLAGE**(
♦ **ID** = id, [**int**] ou [**str**]
♦ **TYP_ENT** = / **'LIGNE'**,
/ **'SURFACE'**,
/ **'VOLUME'**,
♦ **ENT** = ent, [**list<int ou str>**]
) # *fin ASSEMBLAGE*

ID : entier ou chaîne de caractères identifiant l'entité générée par l'assemblage.
TYP_ENT : chaîne de caractères spécifiant le type de l'entité à générer et des entités à assembler parmi **'LIGNE'**, **'SURFACE'** et **'VOLUME'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités à assembler.

2.7.11 Classe BLOC

La classe **BLOC** est la classe permettant de définir un bloc d'entités de même type.

bloc = **BLOC**(
♦ **ID** = id, [**int**] ou [**str**]
♦ **TYP_ENT** = / **'POINT'**,
/ **'LIGNE'**,
/ **'SURFACE'**,
/ **'VOLUME'**,
♦ **ENT** = ent, [**list<int ou str>**]
) # *fin BLOC*

ID : entier ou chaîne de caractères identifiant le bloc d'entité.
TYP_ENT : chaîne de caractères spécifiant le type de l'entité à générer et des entités à assembler parmi **'POINT'**, **'LIGNE'**, **'SURFACE'** et **'VOLUME'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités constitutives du bloc.

2.7.12 Classe EXTRUSION

La classe **EXTRUSION** est la classe permettant de réaliser une extrusion d'entités de même type.

```
extr = EXTRUSION(
        ♦ TYP_ENT = / 'POINT',
                    / 'LIGNE',
                    / 'SURFACE',
        ♦ ENT = ent, [list<int ou str>]
        ◊ / TRANSLATION = trans, [list<float>]
          / ROTATION = rota, [list<float>]
          / DEPLACEMENT = depla, [list<float>]
        ◊ COUCHES = couch, [list<(int, float)>]
        ◊ ENT_OUT = out, [<ENTITES, str>]
        ) # fin EXTRUSION
```

TYP_ENT : chaîne de caractères spécifiant le type des entités à extruder parmi **'POINT'**, **'LIGNE'** et **'SURFACE'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités à extruder.
TRANSLATION : liste de 3 réels donnant les composantes du vecteur de translation.
ROTATION : liste de 7 réels donnant successivement les 3 composantes d'un vecteur portant l'axe de rotation, les 3 coordonnées d'un point sur cet axe, l'angle de la rotation.
DEPLACEMENT : liste de 10 réels donnant les composantes d'un déplacement, composé d'une translation (3 réels) et d'une rotation (7 réels).
COUCHES : liste de couples entier/réel décrivant le découpage en couches à effectuer lors de l'extrusion. Le nombre de couples indique le nombre de couches à générer et le couple entier/réel indique pour chaque couche le nombre de mailles à générer (dans le sens de l'extrusion) et la position finale normalisée de la couche dans le sens de l'extrusion (réel compris entre 0. et 1.).
ENT_OUT : liste de taille 2 donnant une instance de la classe **ENTITES** ainsi qu'une chaîne de caractères permettant de référencer les entités générées par l'extrusion. L'instance de **ENTITES** aura été créée au préalable et renseignée via la méthode **add**. Par exemple, dans le cas de l'extrusion d'une surface, l'affectation des références aux entités générées s'effectue ainsi :
— out['chaine'][0] correspond à la surface sommet générée,
— out['chaine'][1] correspond au volume généré,
— out['chaine'][2], ... correspondent aux surfaces latérales générées (dans l'ordre des lignes constitutives de la boucle ayant servi à créer la surface extrudée).
où out et 'chaine' sont les composantes de **ENT_OUT**.

2.7.13 Classe TRANSFORMATION

La classe **TRANSFORMATION** est la classe permettant de réaliser une transformation d'entités de même type.

```
transfo = TRANSFORMATION(
          ♦ TYP_ENT = / 'POINT',
                      / 'LIGNE',
                      / 'SURFACE',
```

```
                / 'VOLUME',
♦ ENT = ent, [list<int ou str>]
◊ / DILATATION = trans, [list<float>]
  / TRANSLATION = rota, [list<float>]
  / ROTATION = rota, [list<float>]
  / SYMETRIE = depla, [list<float>]
◊ COPIE = / 'OUI',
          / 'NON',
◊ ENT_OUT = out, [<ENTITES, str>]
) # fin TRANSFORMATION
```

TYP_ENT : chaîne de caractères spécifiant le type des entités à transformer parmi **'POINT'**, **'LIGNE'**, **'SURFACE'** et **'VOLUME'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités à transformer.
DILATATION : liste de 4 réels donnant les 3 composantes du vecteur de dilatation et le facteur de dilatation.
TRANSLATION : liste de 3 réels donnant les composantes du vecteur de translation.
ROTATION : liste de 7 réels donnant successivement les 3 composantes d'un vecteur portant l'axe de rotation, les 3 coordonnées d'un point sur cet axe, l'angle de la rotation.
SYMETRIE : liste de 4 réels A, B, C, D donnant les coefficients de l'équation cartésienne du plan de symétrie $Ax + By + Cz + D = 0$.
COPIE : chaîne de caractères spécifiant la copie (**'OUI'**) ou pas (**'NON'**) des entités transformées.
ENT_OUT : liste de taille 2 contenant une instance de la classe **ENTITES** ainsi qu'une chaîne de caractères permettant de référencer les entités générées par la transformation avec copie. L'instance de **ENTITES** aura été créée au préalable et renseignée via la méthode **add**.

2.7.14 Classe STRUCTURE

La classe **STRUCTURE** est la classe permettant de réaliser un maillage structuré (dit "transfinite") d'entités de même type.

```
struc = STRUCTURE(
        ♦ TYP_ENT = / 'LIGNE',
                    / 'SURFACE',
                    / 'VOLUME',
        ♦ ENT = ent, [list<int ou str>]
        ◊ PROG = prog, [/ <int, 'UNI', float>
                       / <int, 'GEOM', float>
                       / <int, 'SAUT', float>]
        ◊ POINTS = points, [list<int>]
        ) # fin STRUCTURE
```

TYP_ENT : chaîne de caractères spécifiant le type des entités à structurer parmi **'LIGNE'**, **'SURFACE'** et **'VOLUME'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités à structurer.

PROG : dans le cas de la structuration d'une entité de type **'LIGNE'**, liste de taille 3 contenant un entier spécifiant le nombre de noeuds à créer sur la ligne, une chaîne de caractères spécifiant le mode de répartition de ces noeuds (uniforme avec **'UNI'**, raffiné selon une progression géométrique avec **'GEOM'**, raffiné au voisinage des extrémités de la ligne avec **'SAUT'**), un réel paramétrisant les modes avec raffinement.
POINTS : dans le cas de la structuration d'une entité de type **'SURFACE'** ou **'VOLUME'**, liste d'entiers ou chaînes de caractères référençant des points situés sur la frontière de l'entité (3 ou 4 points pour une **'SURFACE'**, 6 ou 8 points pour un **'VOLUME'**) et définissant les coins de l'interpolation transfinie à réaliser par la structuration.

2.7.15 Classe RECOMBINE

La classe **RECOMBINE** est la classe permettant de modifier les maillages triangles de surfaces en des maillages mixtes triangles/quadrangles en recombinant les triangles.

recomb = **RECOMBINE**(
♦ **TYP_ENT** = / **'SURFACE'**,
♦ **ENT** = ent, [**list<int ou str>**]
) # *fin RECOMBINE*

TYP_ENT : chaîne de caractères spécifiant le type des entités à recombiner (**'SURFACE'**).
ENT : liste d'entiers ou chaînes de caractères référençant les entités à recombiner.

2.7.16 Classe COHERENCE

La classe **COHERENCE** est la classe permettant de supprimer les entités géométriques dupliquées. Par défaut, cette suppression est réalisée automatiquement après chaque transformation géométrique, mais ceci peut être modifé via la classe **OPTION** (mot-clé **GEOM_AUTO_COHERENCE**).

coher = **COHERENCE**(
) # *fin COHERENCE*

2.7.17 Classe FINESSE

La classe **FINESSE** est la classe permettant de fixer la finesse du maillage au voisinage d'un ensemble de points.

fin = **FINESSE**(
♦ **POINTS** = ent, [**list<int ou str>**]
♦ **H** = h, [**float**]
) # *fin FINESSE*

POINTS : liste d'entiers ou chaînes de caractères référençant des points au voisinage desquels on veut fixer la finesse du maillage.
H : réel donnant la taille caractéristique des éléments caractérisant la finesse souhaitée.

2.7.18 Classe DESTRUCTION

La classe **DESTRUCTION** est la classe permettant de supprimer en ensemble d'entités de même type.

```
destr = DESTRUCTION(
        ♦ TYP_ENT = / 'POINT',
                    / 'LIGNE',
                    / 'SURFACE',
                    / 'VOLUME',
        ♦ ENT = ent, [list<int ou str>]
        ) # fin DESTRUCTION
```

TYP_ENT : chaîne de caractères spécifiant le type des entités à supprimer parmi **'POINT'**, **'LIGNE'**, **'SURFACE'** et **'VOLUME'**.
ENT : liste d'entiers ou chaînes de caractères référençant les entités à supprimer.

2.7.19 Classe OPTION

La classe **OPTION** est la classe permettant de définir des options relatives à la géométrie ou au maillage du domaine.

```
opt = OPTION(
      ◊ GEOM_AUTO_COHERENCE = / 0,
                              / 1,
      ◊ GEOM_TOLERANCE = tol, [float]
      ◊ MAIL_ORDRE_ELEM = / 1,
                          / 2,
      ◊ MAIL_ORDRE_2_INCOMPLET = / 0,
                                 / 1,
      ◊ MAIL_LISSAGE = liss, [int]
      ) # fin OPTION
```

GEOM_AUTO_COHERENCE : entier spécifiant la suppression (**1**, valeur par défaut) ou non (**0**) des entités dupliquées.
GEOM_TOLERANCE : réel spécifiant la tolérance géométrique (1.e-06 par défaut)
MAIL_ORDRE_ELEM : entier spécifiant l'ordre des éléments créés (1 par défaut).
MAIL_ORDRE_2_INCOMPLET : entier spécifiant la création d'éléments du deuxième ordre incomplets (**1**) ou non (**0**, valeur par défaut).
MAIL_LISSAGE : entier spécifiant le nombre de pas de lissage appliqués au maillage final (1 par défaut)

2.7.20 Classe ENTITES

La classe **ENTITES** est utile pour stocker des chaînes de caractères permettant de référencer des entités du domaine lorsqu'on ne connaît pas leur numéro. Par exemple, lorsqu'on extrude (via **EXTRUSION**) ou transforme (via **TRANSFORMATION**) des entités, la numérotation des entités générées est inconnue mais on peut spécifier des chaines de caractères qui permettront de référencer ces entités plus tard.
Cette classe dérive de la classe dictionnaire du langage Python. Son constructeur est vide :

ent = **ENTITES**(
) # *fin ENTITES*

On la renseigne via sa méthode **add** qui ajoute une paire (clé, valeur) au dictionnaire sous-jacent. Par exemple, le code :

```
ent = ENTITES()
ent.add(NOM = 'mes_ent', TAILLE = 2)
```

crée une instance de la classe **ENTITES** puis lui ajoute :

— une clé : la chaine 'mes_ent'

— une valeur : un tableau de taille 2 contenant les chaînes 'mes_ent[0]' et 'mes_ent[1]'

Dans le code suivant, on extrude la ligne de numéro 1 par translation : ceci génère au moins 2 nouvelles entités (une ligne et une surface) qui pourront ensuite être référencées via l'instance ent de **ENTITES** et la clé 'mes_ent' passées via le mot clé **ENT_OUT**.

```
OPER = [ ...,
    EXTRUSION(TYP_ENT = 'LIGNE', ENT = [1], TRANSLATION = [0.,h,0.], ENT_OUT = [ent, 'mes_ent']),
    ...],
```

Dans le code suivant, on crée un bloc de deux lignes composé de la ligne de numéro 1 et de la ligne créée par l'extrusion précédente que l'on référence par ent['mes_ent'][0]. De façon analogue, la surface générée par l'extrusion peut être référencée par ent['mes_ent'][1].

```
OPER = [ ...,
    BLOC(ID = 1, TYP_ENT = 'LIGNE', ENT = [1,ent['mes_ent'][0]]),
    ...],
```

Méthodes

add() : méthode avec deux arguments **NOM** ([**str**]) et **TAILLE** ([**int**]) permettant d'ajouter un ensemble de chaînes de caractères référençant des entités.

2.7.21 Classe ID

La classe **ID** permet de déléguer la numérotation des entités au programme. On fournit juste une chaîne de caractères permettant de référencer l'entité pour des opérations ultérieures.

iden = **ID**(
♦ **TYP_ENT** = / **'POINT'**,
/ **'LIGNE'**,
/ **'SURFACE'**,
/ **'VOLUME'**,
/ **'REGION'**,
♦ **ID** = id, [**str**]
) # *fin ID*

TYP_ENT : chaîne de caractères spécifiant le type de l'entité à identifier parmi **'POINT'**, **'LIGNE'**, **'SURFACE'**, **'VOLUME'** et **'REGION'**.

ID : chaîne de caractères pour référencer l'entité.

Par exemple, dans le code suivant, on délègue la numérotation de deux points en spécifiant les chaînes 'p1' et 'p2' à leur création. Ces deux points peuvent ensuite être référencés pour la création de trois lignes dont on délègue également la numérotation au programme.

```
OPER = [ ... ,
    ID('POINT','p1'), POINT('p1', -0.05, 0.05, 0., lc),
    ID('POINT','p2'), POINT('p2', -0.05, 0.1, 0., lc),
    ID('LIGNE','l1'), LIGNE('l1', 1, 'p1'),
    ID('LIGNE','l2'), LIGNE('l2', 'p1', 'p2'),
    ID('LIGNE','l3'), LIGNE('l3', 'p2', 4),
    ...] ,
```

2.8 Données relatives aux maillages

2.8.1 Classe MAILLAGE

La classe **MAILLAGE** est la classe permettant d'importer ou d'exporter un maillage. Le fichier de maillage (en entrée ou en sortie) est défini via la classe **FICHIER**.
Dans le contexte d'une étude phasée (**PHASAGE**), elle permet également de définir la partition du maillage en groupes au sens de CESAR.

```
mail = MAILLAGE(
        ♦ NOM = nom, [str]
        ◊ FIC_IN = FICHIER(
                    ♦ FORMAT = / 'GMSH',
                               / 'MED',
                    ♦ NOM = nom, [str]
                    ), # fin FICHIER
                    ♦ DIM = / 1,
                            / 2,
                            / 3,
                    ♦ DIM_ESPACE = / 2,
                                   / 3,
        ◊ FIC_OUT = FICHIER(
                      ♦ FORMAT = / 'GMSH',
                                 / 'MED',
                                 / 'ASTER',
                      ♦ NOM = nom, [str]
                      ), # fin FICHIER
        ◊ GRPES = grpes, [list<GROUPE>]
        ) # fin MAILLAGE
```

NOM : chaîne de caractères définissant le nom du maillage. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
FIC_IN : objet de type **FICHIER** définissant le fichier de maillage à importer. Actuellement, les formats **'GMSH'** et **'MED'** sont disponibles.
DIM : entier définissant la dimension du maillage : **1** (resp. **2**, **3**) pour un maillage linéique (resp. surfacique, volumique).
DIM_ESPACE : entier définissant la dimension de l'espace ambiant : **2** (resp. **3**) pour le plan (resp. l'espace).
FIC_OUT : objet de type **FICHIER** définissant le fichier de maillage à exporter. Actuellement, les formats **'GMSH'**, **'MED'** et **'ASTER'** sont disponibles.

GRPES : liste d'objets de type **GROUPE** donnant, dans un contexte de phasage de construction, la partition du maillage en groupes au sens de CESAR.

Autres attributs

mesh : objet de type **GMESH** stockant le maillage au format MED Mémoire. De façon standard, l'utilisateur construit son maillage à l'aide d'un outil externe (GMSH ou SALOME pour l'instant) et définit le fichier de maillage à importer via l'attribut **FIC_IN**. Dans ce cas, l'attribut **mesh** est renseigné en interne par le Pilote. Mais l'utilisateur a également la possibilité de définir le maillage "à la main" en renseignant lui-même l'attribut **mesh**. Ceci se fait en utilisant le langage de la librairie MED Mémoire dont l'objet **mesh** relève. Les tests zzzz001a et zzzz001b montrent un tel exemple d'utilisation.

Méthodes

importer() : méthode sans paramètres permettant d'importer le maillage depuis un fichier défini par l'attribut **FIC_IN**.
exporter() : méthode sans paramètres permettant d'exporter le maillage vers un fichier défini par l'attribut **FIC_OUT**. Les tests zzzz001a et zzzz001b montrent un exemple d'utilisation.
convertir() : méthode sans paramètres permettant de convertir un fichier de maillage défini par l'attribut **FIC_IN** vers un fichier de maillage défini par l'attribut **FIC_OUT**. Les tests zzzz002 et zzzz003 montrent un exemple d'utilisation.
extraire() : méthode permettant d'extraire du maillage les coordonnées des noeuds d'un **LIEU**. Elle possède 2 arguments à passer via les 2 mots-clés suivants :

. **GRANDEUR** : le champ de coordonnées, de type **COOR_2D** ou **COOR_3D**, à extraire.
. **LIEU** : le lieu de l'extraction.

Elle renvoie un objet de type **EXTRACT** contenant une donnée **val** qui est un dictionnaire contenant le champ de coordonnées.

Exemples

Exemple de définition du maillage par import d'un fichier GMSH, dans le cas d'une étude de plaque (maillage de dimension 2 et espace de dimension 3).

```
from modele_donnees import *

rep_trav = 'tests/atelier/'
fic_msh = rep_trav + 'ssls002a.msh'

mail = MAILLAGE(NOM = 'mon_mail',
                FIC_IN = FICHIER(NOM = fic_msh,
                                 FORMAT = 'GMSH'),
                DIM = 2,
                DIM_ESPACE = 3)
```

Exemple de définition manuelle du maillage et d'export au format GMSH. Pour utiliser le langage MED Mémoire, on doit importer en préambule le module Python libMEDMEM_Swig.

```
from modele_donnees import *
from libMEDMEM_Swig import *

rep_trav = 'tests/atelier/'
fic_msh = rep_trav + 'zzzz001a.msh'


#definiton manuelle de l'objet mesh
#==================================
```

```
meshing = MESHING()
meshing.setName('mon_mail')

#noeuds
#------

coordo = [0.0, 0.0, 0.0  ,
          0.0, 0.0, 1.0  ,
          2.0, 0.0, 1.0  ,
          0.0, 2.0, 1.0  ,
          -2.0, 0.0, 1.0 ,
          0.0, -2.0, 1.0 ,
          1.0, 1.0, 2.0  ,
          -1.0, 1.0, 2.0 ,
          -1.0, -1.0, 2.0,
          1.0, -1.0, 2.0 ,
          1.0, 1.0, 3.0  ,
          -1.0, 1.0, 3.0 ,
          -1.0, -1.0, 3.0,
          1.0, -1.0, 3.0 ,
          1.0, 1.0, 4.0  ,
          -1.0, 1.0, 4.0 ,
          -1.0, -1.0, 4.0,
          1.0, -1.0, 4.0 ,
          0.0, 0.0, 5.0]
meshing.setCoordinates(3, 19, coordo, 'CARTESIAN', MED_FULL_INTERLACE)

#mailles
#-------

entite = MED_CELL

#nombre et types de mailles
meshing.setNumberOfTypes(3, MED_CELL)
meshing.setTypes([MED_TETRA4, MED_PYRA5, MED_HEXA8], MED_CELL)
meshing.setNumberOfElements([12, 2, 2], MED_CELL)

#connectivites
connec_tetra =  [1,2,3,6,
                 1,2,4,3 ,
                 1,2,5,4 ,
                 1,2,6,5 ,
                 2,7,4,3 ,
                 2,8,5,4 ,
                 2,9,6,5 ,
                 2,10,3,6,
                 2,7,3,10,
                 2,8,4,7 ,
                 2,9,5,8 ,
                 2,10,6,9]
connec_pyra = [7,8,9,10,2,
               15,18,17,16,19]
connec_hexa = [11,12,13,14,7,8,9,10,
               15,16,17,18,11,12,13,14]
meshing.setConnectivity(entite, MED_TETRA4, connec_tetra)
meshing.setConnectivity(entite, MED_PYRA5, connec_pyra)
meshing.setConnectivity(entite, MED_HEXA8, connec_hexa)

#faces
#-----

#nombre et types de faces
meshing.setNumberOfTypes(2, MED_FACE)
meshing.setTypes([MED_TRIA3, MED_QUAD4], MED_FACE)
meshing.setNumberOfElements([4, 4], MED_FACE)

#connectivites
connec_tria = [1,4,3,
               1,5,4,
               1,6,5,
               1,3,6]
connec_quad = [7,8,9,10,
               11,12,13,14,
               11,7,8,12  ,
               12,8,9,13]
```

```
meshing.setConnectivity(MED_FACE, MED_TRIA3, connec_tria)
meshing.setConnectivity(MED_FACE, MED_QUAD4, connec_quad)


#groupes de noeuds
#-----------------

grpe = GROUP()
grpe.setName("Des_noeuds")
grpe.setMesh(meshing)
grpe.setEntity(MED_NODE)
grpe.setNumberOfGeometricType(1)
grpe.setGeometricType([MED_NONE])
grpe.setNumberOfElements([4])
grpe.setNumber([1,5], [1,4,5,7])
meshing.addGroup(grpe)

grpe = GROUP()
grpe.setName("Autres_noeuds")
grpe.setMesh(meshing)
grpe.setEntity(MED_NODE)
grpe.setNumberOfGeometricType(1)
grpe.setGeometricType([MED_NONE])
grpe.setNumberOfElements([3])
grpe.setNumber([1,4], [2,3,6])
meshing.addGroup(grpe)

#groupes de mailles
#-----------------

grpe = GROUP()
grpe.setName("Des_mailles")
grpe.setMesh(meshing)
grpe.setEntity(MED_CELL)
grpe.setNumberOfGeometricType(3)
grpe.setGeometricType([MED_TETRA4, MED_PYRA5, MED_HEXA8])
grpe.setNumberOfElements([4,1,2])
grpe.setNumber([1,5,6,8],[2,7,8,12,13,15,16])
meshing.addGroup(grpe)

grpe = GROUP()
grpe.setName("Autres_mailles")
grpe.setMesh(meshing)
grpe.setEntity(MED_CELL)
grpe.setNumberOfGeometricType(2)
grpe.setGeometricType([MED_TETRA4, MED_PYRA5])
grpe.setNumberOfElements([4,1])
grpe.setNumber([1,5,6],[3,4,5,9,14])
meshing.addGroup(grpe)

#groupes de faces
#----------------

grpe = GROUP()
grpe.setName("Des_faces")
grpe.setMesh(meshing)
grpe.setEntity(MED_FACE)
grpe.setNumberOfGeometricType(2)
grpe.setGeometricType([MED_TRIA3,MED_QUAD4])
grpe.setNumberOfElements([2,3])
grpe.setNumber([1,3,6],[2,4,5,6,8])
meshing.addGroup(grpe)

grpe = GROUP()
grpe.setName("Autres_faces")
grpe.setMesh(meshing)
grpe.setEntity(MED_FACE)
grpe.setNumberOfGeometricType(1)
grpe.setGeometricType([MED_TRIA3])
grpe.setNumberOfElements([2])
grpe.setNumber([1,3],[1,3])
meshing.addGroup(grpe)


#creation du maillage et affectation de l'objet mesh
```

```
#=============================================
mail = MAILLAGE(NOM = 'mon_mail',
                FIC_OUT = FICHIER(NOM = fic_msh,
                                  FORMAT = 'GMSH'))
mail.setMesh(meshing)


#export du maillage
#=============================
mail.exporter()
```

Exemple d'import du maillage au format GMSH et de conversion au format ASTER.

```
from modele_donnees import *

rep_trav = 'tests/atelier/'
fic_msh = rep_trav + 'zzzz002.msh'
fic_mail = rep_trav + 'zzzz002.mail'


fic_1 = FICHIER(NOM = fic_msh,
                FORMAT = 'GMSH')

fic_2 = FICHIER(NOM = fic_mail,
                FORMAT = 'ASTER')

mail = MAILLAGE(NOM = 'mon_mail',
                FIC_IN = fic_1,
                FIC_OUT = fic_2,
                DIM = 2,
                DIM_ESPACE = 2)

mail.convertir()
```

Exemple d'extraction et d'affichage des coordonnées du maillage.

```
coor = mail.extraire(GRANDEUR = COOR_3D(X = 'OUI',
                                        Y = 'OUI',
                                        Z = 'OUI'),
                     LIEU = LIEU(NOMS = ['Des_mailles'],
                                 TYP = 'GROUP_MAILLE'))

for noeud in coor.val:
    print coor.val[noeud]['X'], coor.val[noeud]['Y'], coor.val[noeud]['Z']
```

2.8.2 Classe GMESH

La classe **GMESH** est la classe dont relève l'attribut **mesh** de la classe **MAILLAGE**. Elle fait partie de l'API Python de la librairie MED Mémoire dont on trouve la documentation dans le répertoire prérequis du répertoire d'installation du Pilote. Pour la consulter, ouvrir par exemple les fichiers suivants avec un navigateur html :

- `prerequis/MED/share/doc/salome/gui/MED/medmem.html` pour MED Mémoire,
- `prerequis/MED/share/doc/salome/gui/MED/classMEDMEM_1_1GMESH.html` pour la classe **GMESH** en particulier.

2.8.3 Classe GROUPE

Le solveur CESAR impose dans son interface que les éléments du maillage soient partitionnés en 1 ou plusieurs "groupes", un groupe devant réunir des éléments relevant de la même famille (par exemple des éléments de mécanique bidimensionnelle ou bien des éléments de coque, ...) indépendamment de leur topologie (par exemple un groupe peut contenir aussi bien des triangles que des quadrangles, ...), et possédant les mêmes caractéristiques de matériau.

Cette logique n'est pas reprise dans le langage Pilote, qui n'impose pas à l'utilisateur de constituer de tels groupes : au contraire, l'utilisateur affecte de façon indépendante (mais néanmoins cohérente !) des propriétés de **MODELE**, **MATERIAU** ou de **CARACTERISTIQUE**, et c'est le Pilote qui se charge de générer lui-même les groupes au sens de CESAR pour le fichier .data.
Néanmoins, dans le cas de l'étude d'un phasage de construction, il est essentiel et naturel que l'utilisateur définisse dans ses données les parties de la structure qui sont présentes ou absentes lors des différentes phases, comme il doit le faire lorsqu'il écrit un .data pour CESAR. Dans ce cas, le langage Pilote introduit la notion de groupe via la classe **GROUPE**, dérivée de **LIEU**, qui permet ainsi à l'utilisateur de définir lui-même les groupes d'éléments du maillage au sens de CESAR.

```
grpe = GROUPE(
        ♦ NOMS = noms, [list<str>]
        ♦ TYP = / 'GROUP_MAILLE',
                / 'GROUP_FACE',
                / 'GROUP_ARETE',
                / 'MAILLE',
                / 'FACE',
                / 'ARETE',
        ♦ NOM = nom, [str]
        ♦ / FORMUL = / 'MECA_2D_DPLAN',
                     / 'MECA_2D_CPLAN',
                     / 'MECA_3D',
                     / 'POUT_2D',
                     / 'POUT_3D',
                     / 'POUT_3D_MULTI',
                     / 'COQ_MINC',
                     / 'COQ_EPAIS',
                     / 'COQ_MULTI',
                     / 'BAR_2D',
                     / 'BAR_3D',
                     / 'AXISYM',
          / LIAISON = / 'CONTACT',
                     / 'JOINT',
                     / 'RELA_LINE',
                       ♦ NB_NOEUD_ELEM = nb_noeud_elem, [int]
                     / 'MATR_ELEM',
                       ♦ NB_NOEUD_ELEM = nb_noeud_elem, [int]
                       ♦ NB_DDL_NOEUD = nb_ddl_noeud, [int]
                       ♦ MATR = / 'RIGI',
                                / 'MASS',
                                / 'AMOR',
        ) # fin GROUPE
```

NOMS : liste de chaînes de caractères définissant des noms d'entités de lieu.
TYP : chaîne de caractères définissant le type des entités de lieu, à choisir parmi :
— **'GROUP_MAILLE'** : pour des groupes de mailles du maillage,
— **'GROUP_FACE'** : pour des groupes de faces du maillage,
— **'GROUP_ARETE'** : pour des groupes d'arêtes du maillage,

— **'MAILLE'** : pour des mailles du maillage,
— **'FACE'** : pour des faces du maillage,
— **'ARETE'** : pour des arêtes du maillage,
NOM : chaîne de caractères définissant le nom du groupe.
FORMUL : chaîne de caractères permettant de définir la formulation commune des éléments finis du groupe, à choisir parmi :
— **'MECA_2D_DPLAN'** : pour la formulation de mécanique bidimensionnelle en déformations planes,
— **'MECA_2D_CPLAN'** : pour la formulation de mécanique bidimensionnelle en contraintes planes,
— **'MECA_3D'** : pour la formulation de mécanique tridimensionnelle ,
— **'POUT_2D'** : pour la formulation de poutre bidimensionnelle,
— **'POUT_3D'** : pour la formulation de poutre tridimensionnelle ,
— **'POUT_3D_MULTI'** : pour la formulation de poutre tridimensionnelle multi-fibres,
— **'COQ_MINC'** : pour la formulation de coque mince,
— **'COQ_EPAIS'** : pour la formulation de coque épaisse,
— **'COQ_MULTI'** : pour la formulation de coque épaisse multi-couches,
— **'BAR_2D'** : pour la formulation de barre bidimensionnelle,
— **'BAR_3D'** : pour la formulation de barre tridimensionnelle,
— **'AXISYM'** : pour la formulation de mécanique bidimensionnelle axisymétrique.
LIAISON : chaîne de caractères permettant de définir le type de liaison concernant les éléments finis du groupe, à choisir parmi :
— **'CONTACT'** : pour définir une zone de contact,
— **'JOINT'** : pour définir une zone de joint,
— **'RELA_LINE'** : pour définir une relation linéaire entre degrés de liberté,
— **'MATR_ELEM'** : pour définir une matrice élémentaire.
NB_NOEUD_ELEM : entier donnant le nombre de noeuds par élément.
NB_DDL_NOEUD : entier donnant le nombre de degrés de liberté par noeud.
MATR : chaîne de caractères permettant de définir le type de matrice élémentaire, à choisir parmi :
— **'RIGI'** : pour définir une matrice de rigidité,
— **'MASS'** : pour définir une matrice de masse,
— **'AMOR'** : pour définir une matrice d'amortissement.

2.8.4 Classe COOR_2D

La classe **COOR_2D** est la classe permettant de définir les composantes du champ de coordonnées bidimensionnelles à extraire.

```
coor2d = COOR_2D(
        ♦ X = / 'OUI',
              / 'NON',
        ♦ Y = / 'OUI',
              / 'NON',
        ) # fin COOR_2D
```

U : coordonnée X dans le repère du maillage.
V : coordonnée Y dans le repère du maillage.

2.8.5 Classe COOR_3D

La classe **COOR_3D** est la classe permettant de définir les composantes du champ de coordonnées tridimensionnelles à extraire.

```
coor3d = COOR_3D(
        ♦ X = / 'OUI',
              / 'NON',
        ♦ Y = / 'OUI',
              / 'NON',
        ♦ Z = / 'OUI',
              / 'NON',
        ) # fin COOR_3D
```

U : coordonnée X dans le repère du maillage.
V : coordonnée Y dans le repère du maillage.
W : coordonnée Z dans le repère du maillage.

2.9 Données relatives aux modèles

2.9.1 Classe MODELE

La classe **MODELE** est la classe permettant de définir les formulations des éléments finis utilisés. Par exemple : éléments finis de mécanique tridimensionnelle, éléments finis de coque mince de Kirchhoff, éléments finis de poutre de Timoshenko, ...

```
mod = MODELE(
        ♦ NOM = nom, [str]
        ♦ MAIL = mail, [MAILLAGE]
        ♦ AFFECT = affect, [list<AFFECT>]
        ) # fin MODELE
```

NOM : chaîne de caractères définissant le nom du modèle. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'applique le modèle.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de modèle. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

```
affect = [ # debut liste
          AFFECT(
          ♦ PROPRIETE = formul, [FORMULATION]
          ♦ LIEU = LIEU(
                   ♦ TYP = / 'GROUP_MAILLE',
                           / 'MAILLE',
                   ♦ NOMS = noms, [list<str>]
                   ), # fin LIEU
          ), # fin AFFECT
        ] # fin liste
```

Ainsi, dans le contexte de la classe **MODELE** :
- la propriété doit être un objet de type **FORMULATION**,
- la propriété ne peut être appliquée que sur des **'MAILLE'** ou des **'GROUP_MAILLE'**.

Néanmoins, la classe **FORMULATION** est une classe abstraite et on ne l'instancie pas. On invoque en fait les classes instanciables suivantes qui en dérivent :
- **FORMUL_MECA**,
- **FORMUL_DIFFUS**,
- **FORMUL_COUPLAG**.

2.9.2 Classe FORMUL_MECA

La classe **FORMUL_MECA** permet de définir une formulation pour un problème de Mécanique.

```
fomeca = FORMUL_MECA(
        ♦ TYP = / 'MECA_2D_DPLAN',
                / 'MECA_2D_CPLAN',
                / 'MECA_3D',
                / 'BAR_2D',
                / 'BAR_2D_FROT',
                / 'BAR_3D',
                / 'BAR_3D_FROT',
                / 'POUT_2D',
                / 'POUT_3D',
                / 'POUT_3D_MULTI',
                / 'COQ_MINC',
                / 'COQ_EPAIS',
                / 'COQ_MULTI',
                / 'AXISYM',
        ) # fin FORMUL_MECA
```

TYP : chaîne de caractère caractérisant la formulation des éléments finis utilisés :
- **MECA_2D_DPLAN** : pour la formulation de mécanique bidimensionnelle en déformations planes,
- **MECA_2D_CPLAN** : pour la formulation de mécanique bidimensionnelle en contraintes planes,
- **MECA_3D** : pour la formulation de mécanique tridimensionnelle,
- **BAR_2D** : pour la formulation de barre plane,
- **BAR_2D_FROT** : pour la formulation de barre plane frottante,
- **BAR_3D** : pour la formulation de barre tridimensionnelle,
- **BAR_3D_FROT** : pour la formulation de barre tridimensionnelle frottante,
- **POUT_2D** : pour la formulation de poutre épaisse plane,
- **POUT_3D** : pour la formulation de poutre épaisse tridimensionnelle (Timoshenko),
- **POUT_3D_MULTI** : pour la formulation de poutre multi-fibres,
- **COQ_MINC** : pour la formulation de coque mince (Kirchhoff),
- **COQ_EPAIS** : pour la formulation de coque épaisse (Reissner),
- **COQ_MULTI** : pour la formulation de coque multi-couches,
- **AXISYM** : pour la formulation de mécanique bidimensionnelle en axisymétrie.

2.9.3 Classe FORMUL_DIFFUS

La classe **FORMUL_DIFFUS** permet de définir une formulation pour un problème de Diffusion.

```
fodiffu = FORMUL_DIFFUS(
        ♦ TYP = / 'DIFFUS_2D',
                / 'ECHANG_2D',
        ) # fin FORMUL_DIFFUS
```

TYP : chaîne de caractère caractérisant la formulation des éléments finis utilisés :
- **DIFFUS_2D** : pour la formulation de diffusion bidimensionnelle,
- **DIFFUS_3D** : pour la formulation de diffusion tridimensionnelle,
- **ECHANG_2D** : pour la formulation d'échange en diffusion bidimensionnelle,
- **ECHANG_3D** : pour la formulation d'échange en diffusion tridimensionnelle.

2.9.4 Classe FORMUL_COUPLAG

Non implémenté à ce jour.

2.10 Données relatives aux matériaux

2.10.1 Classe MATERIAU

La classe **MATERIAU** est la classe permettant de définir les lois de comportement des matériaux à affecter aux éléments finis utilisés.
Par exemple : loi de type élasticité linéaire isotrope ou orthotrope, ou loi non linéaire telle que la loi Mohr-Coulomb sans écrouissage, la loi Von-Mises avec écrouissage, ...
Elle permet également de définir des matériaux renforcés ou composites (pour les poutres multi-fibres ou les coques multi-couches).

```
mat = MATERIAU(
        ♦ NOM = nom, [str]
        ♦ MAIL = mail, [MAILLAGE]
        ♦ AFFECT = affect, [list<AFFECT>]
        ) # fin MATERIAU
```

NOM : chaîne de caractères définissant le nom du matériau. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'applique le matériau.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de matériau. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

```
affect = [ # debut liste
          AFFECT(
          ♦ PROPRIETE = comport, [COMPORTEMENT]
          ♦ LIEU = LIEU(
                   ♦ TYP = / 'GROUP_MAILLE',
```

```
                    / 'MAILLE',
                ♦ NOMS = noms, [list<str>]
                ), # fin LIEU
  ), # fin AFFECT
 ] # fin liste
```

Ainsi, dans le contexte de la classe **MODELE** :
- la propriété doit être un objet de type **COMPORTEMENT**,
- la propriété ne peut être appliquée que sur des **'MAILLE'** ou des **'GROUP_MAILLE'**.

Néanmoins, la classe **COMPORTEMENT** est une classe abstraite et on ne l'instancie pas. On invoque en fait les classes instanciables suivantes qui en dérivent :
- **ELAS_LINE_ISO**,
- **ELAS_LINE_ORTHO**,
- **BETON_JEUNE_AGE**,
- **MOHR_COUL_SANS_ECROU**,
- **VON_MISES_SANS_ECROU**,
- **VON_MISES_AVEC_ECROU**,
- **DRUCK_PRAG_SANS_ECROU**,
- **DRUCK_PRAG_AVEC_ECROU**,
- **CRIT_PARABOL**,
- **VERMEER**,
- **CAM_CLAY_MODIF**,
- **PREVOST_HOEG**,
- **CRIT_ORIENT**,
- **HOEK_BROWN**,
- **TRESCA_ANISO**,
- **ELAS_DILAT_ISO**,
- **ASTER_ELAS**,
- **ASTER_VMIS_ISOT_LINE**,
- **MATER_RENFORCE**,
- **COMPOSITE**,
- **ELASTO_PLAST_1D**,
- **ASTER_ELAS**,
- **ASTER_ELAS_ORTH**,
- **ASTER_VMIS_ISOT_LINE**,
- **DIFFUS_LINE_2D**,
- **ECHANG_LINE**.

2.10.2 Classe ELAS_LINE_ISO

La classe **ELAS_LINE_ISO** permet de définir les caractéristiques de la loi de comportement de type élasticité linéaire isotrope.

```
eli = ELAS_LINE_ISO(
    ♦ NOM = nom, [str]
    ♦ RO = ro, [float]
    ♦ YOUNG = young, [float]
    ♦ POISS = poiss, [float]
    ) # fin ELAS_LINE_ISO
```

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.

2.10.3 Classe ELAS_LINE_ORTHO

La classe **ELAS_LINE_ORTHO** permet de définir les caractéristiques de la loi de comportement de type élasticité linéaire orthotrope.

elo = **ELAS_LINE_ORTHO**(
- ♦ **NOM** = nom, [**str**]
- ♦ **RO** = ro, [**float**]
- ♦ **E1** = e1, [**float**]
- ♦ **E2** = e2, [**float**]
- ♦ **P1** = p1, [**float**]
- ♦ **P2** = p2, [**float**]
- ♦ **G2** = g2, [**float**]
- ♦ **TETA** = teta, [**float**]
- ◇ **PHI** = phi, [**float**]

) # *fin ELAS_LINE_ORTHO*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young dans la direction 1.
E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant l'angle entre l'axe Ox et la direction 1.
PHI : dans le cas 3D, réel donnant l'angle entre la direction 1 et l'axe d'orthotropie.

2.10.4 Classe BETON_JEUNE_AGE

La classe **BETON_JEUNE_AGE** permet de définir les caractéristiques de la loi de comportement du béton au jeune âge.

bja = **BETON_JEUNE_AGE**(
- ♦ **NOM** = nom, [**str**]
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **DILAT** = dilat, [**float**]
- ♦ **RETRA** = retra, [**float**]
- ♦ **SEUIL** = seuil, [**float**]
- ♦ **HYD** = hyd, [**float**]

) # *fin BETON_JEUNE_AGE*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
DILAT : réel donnant la valeur du coefficient de dilatation thermique.
RETRA : réel donnant la valeur du retrait endogène final du béton.
SEUIL : réel donnant la valeur du seuil du matériau durci.
HYD : réel donnant la valeur du degré d'hydratation.

2.10.5 Classe MOHR_COUL_SANS_ECROU

La classe **MOHR_COUL_SANS_ECROU** permet de définir les caractéristiques de la loi de comportement de type Mohr Coulomb sans écrouissage.

mcse = **MOHR_COUL_SANS_ECROU**(
- ♦ **NOM** = nom, [**str**]
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]

) # *fin MOHR_COUL_SANS_ECROU*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la cohésion.
PHI : réel donnant l'angle de frottement interne.
PSI : réel donnant l'angle de dilatance.

2.10.6 Classe VON_MISES_SANS_ECROU

La classe **VON_MISES_SANS_ECROU** permet de définir les caractéristiques de la loi de comportement de type Von-Mises sans écrouissage.

vmse = **VON_MISES_SANS_ECROU**(
- ♦ **NOM** = nom, [**str**]
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **K** = k, [**float**]

) # *fin VON_MISES_SANS_ECROU*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en cisaillement simple.

2.10.7 Classe VON_MISES_AVEC_ECROU

La classe **VON_MISES_AVEC_ECROU** permet de définir les caractéristiques de la loi de comportement de type Von-Mises avec écrouissage.

vmae = **VON_MISES_AVEC_ECROU**(
- **NOM** = nom, [**str**]
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **K** = k, [**float**]
- **H** = h, [**float**]

) # *fin VON_MISES_AVEC_ECROU*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en cisaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.

2.10.8 Classe DRUCK_PRAG_SANS_ECROU

La classe **DRUCK_PRAG_SANS_ECROU** permet de définir les caractéristiques de la loi de comportement de type Drucker-Prager sans écrouissage.

dpse = **DRUCK_PRAG_SANS_ECROU**(
- **NOM** = nom, [**str**]
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **C** = c, [**float**]
- **PHI** = phi, [**float**]
- **PSI** = psi, [**float**]

) # *fin DRUCK_PRAG_SANS_ECROU*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.

2.10.9 Classe DRUCK_PRAG_AVEC_ECROU

La classe **DRUCK_PRAG_AVEC_ECROU** permet de définir les caractéristiques de la loi de comportement de type Drucker-Prager avec écrouissage.

dpae = **DRUCK_PRAG_AVEC_ECROU**(
♦ **NOM** = nom, [**str**]
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **XHI** = xhi, [**float**]
) # *fin DRUCK_PRAG_AVEC_ECROU*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
XHI : réel donnant la valeur du paramètre d'écrouissage

2.10.10 Classe CRIT_PARABOL

La classe **CRIT_PARABOL** permet de définir les caractéristiques de la loi de comportement de type critère parabolique.

cp = **CRIT_PARABOL**(
♦ **NOM** = nom, [**str**]
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **RC** = rc, [**float**]
♦ **RT** = rt, [**float**]
) # *fin CRIT_PARABOL*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple.

2.10.11 Classe VERMEER

La classe **VERMEER** permet de définir les caractéristiques de la loi de comportement de type Vermeer.

ve = **VERMEER**(
♦ **NOM** = nom, [**str**]

♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **EPSO** = epso, [**float**]
♦ **PHICV** = phicv, [**float**]
♦ **PHIP** = phip, [**float**]
♦ **BETA** = beta, [**float**]
♦ **EPSCO** = epsco, [**float**]
♦ **PO** = po, [**float**]
) # *fin VERMEER*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
EPSO : réel donnant la valeur de la déformation volumique élastique initiale.
PHICV : réel donnant la valeur de l'angle de frottement à l'état critique.
PHIP : réel donnant la valeur de l'angle de frottement au pic.
BETA : réel donnant la valeur du paramètre du modèle.
EPSCO : réel donnant la valeur du paramètre du modèle.
PO : réel donnant la valeur de la pression de référence.

2.10.12 Classe CAM_CLAY_MODIF

La classe **CAM_CLAY_MODIF** permet de définir les caractéristiques de la loi de comportement de type Cam Clay modifié.

ccm = **CAM_CLAY_MODIF**(
♦ **NOM** = nom, [**str**]
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **ALOE** = aloe, [**float**]
♦ **AKOE** = akoe, [**float**]
♦ **AMC** = amc, [**float**]
♦ **OED** = oed, [**float**]
♦ **PCO** = pco, [**float**]
) # *fin CAM_CLAY_MODIF*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
ALOE : réel donnant la valeur de la pente de la courbe de consolidation vierge.
AKOE : réel donnant la valeur de la pente des courbes charge-décharge.
AMC : réel donnant la valeur de la pente de la courbe d'état critique.
OED : réel donnant la valeur de l'indice des vides initial.
PCO : réel donnant la valeur de la pression de préconsolidation initiale.

2.10.13 Classe PREVOST_HOEG

La classe **PREVOST_HOEG** permet de définir les caractéristiques de la loi de comportement de type Prévost-Hoeg.

ph = **PREVOST_HOEG**(
♦ **NOM** = nom, [**str**]
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **AO** = ao, [**float**]
♦ **BO** = bo, [**float**]
) # *fin PREVOST_HOEG*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
AO : réel donnant la valeur du paramètre du modèle.
BO : réel donnant la valeur du paramètre du modèle.

2.10.14 Classe CRIT_ORIENT

La classe **CRIT_ORIENT** permet de définir les caractéristiques de la loi de comportement de type critère orienté.

co = **CRIT_ORIENT**(
♦ **NOM** = nom, [**str**]
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ / **ALPHA** = alpha, [**float**]
/ **U** = u, [**list<float>**]
) # *fin CRIT_ORIENT*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
ALPHA : dans le cas 2D, réel donnant la valeur de l'angle par rapport à l'axe Ox.
U : dans le casd 3D, liste de 3 réels donnant les coordonnées du vecteur normal au plan de discontinuité.

2.10.15 Classe HOEK_BROWN

La classe **HOEK_BROWN** permet de définir les caractéristiques de la loi de comportement de type Hoek-Brown.

hb = **HOEK_BROWN**(
- **NOM** = nom, [**str**]
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **SU** = su, [**float**]
- **S** = s, [**float**]
- **M** = m, [**float**]

) # *fin HOEK_BROWN*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
SU : réel donnant la valeur de la contrainte à la rupture en compression de la roche saine.
S : réel donnant la valeur du coefficient de fracturation.
M : réel donnant la valeur du paramètre de forme.

2.10.16 Classe TRESCA_ANISO

La classe **TRESCA_ANISO** permet de définir les caractéristiques de la loi de comportement de type Tresca anisotrope.

ta = **TRESCA_ANISO**(
- **NOM** = nom, [**str**]
- **RO** = ro, [**float**]
- **E1** = e1, [**float**]
- **E2** = e2, [**float**]
- **P1** = p1, [**float**]
- **P2** = p2, [**float**]
- **G2** = g2, [**float**]
- **TETA** = teta, [**float**]
- **A1** = a1, [**float**]
- **B1** = b1, [**float**]
- **A2** = a2, [**float**]
- **B2** = b2, [**float**]

) # *fin TRESCA_ANISO*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young dans la direction 1.
E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.

G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
A1 : réel donnant la valeur de a_1 dans l'expression $a_1 x_2 + b_1$ de la variation de la cohésion en compression dans la direction 2.
B1 : réel donnant la valeur de b_1 dans l'expression $a_1 x_2 + b_1$ de la variation de la cohésion en compression dans la direction 2.
A2 : réel donnant la valeur de a_2 dans l'expression $a_2 x_2 + b_2$ de la variation de la cohésion en extension dans la direction 2.
B2 : réel donnant la valeur de b_2 dans l'expression $a_2 x_2 + b_2$ de la variation de la cohésion en extension dans la direction 2.

2.10.17 Classe ELAS_DILAT_ISO

La classe **ELAS_DILAT_ISO** permet de définir les caractéristiques de la loi de comportement de type élasticité avec dilatance isotrope.

edi = **ELAS_DILAT_ISO**(
♦ **NOM** = nom, [**str**]
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **CKA** = cka, [**float**]
♦ **CKB** = ckb, [**float**]
♦ **CKC** = ckc, [**float**]
) # *fin ELAS_DILAT_ISO*

NOM : chaîne de caractères définissant le nom de la loi.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
CKA : réel donnant la valeur du coefficient K_a affecté au cisaillement.
CKB : réel donnant la valeur du coefficient K_b affecté à la variation de volume.
CKC : réel donnant la valeur du coefficient K_c affecté au troisième invariant.

2.10.18 Classe ASTER_ELAS

La classe **ASTER_ELAS** désigne la loi nommée "ELAS" dans le langage Aster. Elle permet de définir les caractéristiques de la loi de comportement de type élasticité linéaire isotrope pour un calcul avec Aster.

aeli = **ASTER_ELAS**(
♦ **NOM** = nom, [**str**]
♦ **RHO** = rho, [**float**]
♦ **E** = e, [**float**]
♦ **NU** = nu, [**float**]
◊ **AMOR_ALPHA** = amor_alpha, [**float**]
◊ **AMOR_BETA** = amor_beta, [**float**]
) # *fin ASTER_ELAS*

NOM : chaîne de caractères définissant le nom de la loi.
RHO : réel donnant la valeur de la masse volumique.
E : réel donnant la valeur du module d'Young.
NU : réel donnant la valeur du coefficient de Poisson.
AMOR_ALPHA : coefficient d'amortissement de Rayleigh relatif à la matrice de rigidité.
AMOR_BETA : coefficient d'amortissement de Rayleigh relatif à la matrice de masse.

2.10.19 Classe ASTER_ELAS_ORTH

La classe **ASTER_ELAS_ORTH** désigne la loi nommée "ELAS_ORTH" dans le langage Aster. Elle permet de définir les caractéristiques de la loi de comportement de type élasticité linéaire orthotrope pour un calcul avec Aster.

aelo = **ASTER_ELAS_ORTH**(
- ♦ **NOM** = nom, [**str**]
- ♦ **RHO** = rho, [**float**]
- ♦ **E_L** = e_l, [**float**]
- ♦ **E_T** = e_t, [**float**]
- ♦ **E_N** = e_n, [**float**]
- ♦ **G_LT** = g_lt, [**float**]
- ♦ **G_TN** = g_tn, [**float**]
- ♦ **G_LN** = g_ln, [**float**]
- ♦ **NU_LT** = nu_lt, [**float**]
- ♦ **NU_TN** = nu_tn, [**float**]
- ♦ **NU_LN** = nu_ln, [**float**]

) # *fin ASTER_ELAS_ORTH*

NOM : chaîne de caractères définissant le nom de la loi.
RHO : réel donnant la valeur de la masse volumique.
E_L : réel donnant la valeur du module d'Young longitudinal.
E_T : réel donnant la valeur du module d'Young transversal.
E_N : réel donnant la valeur du module d'Young normal.
G_LT : réel donnant la valeur du module de cisaillement dans le plan LT.
G_TN : réel donnant la valeur du module de cisaillement dans le plan TN.
G_LN : réel donnant la valeur du module de cisaillement dans le plan LN.
NU_LT : réel donnant la valeur du coefficient de Poisson dans le plan LT.
NU_TN : réel donnant la valeur du coefficient de Poisson dans le plan TN.
NU_LN : réel donnant la valeur du coefficient de Poisson dans le plan LN.

2.10.20 Classe ASTER_VMIS_ISOT_LINE

La classe **ASTER_VMIS_ISOT_LINE** désigne la loi nommée "VMIS_ISOT_LINE" dans le langage Aster. Elle permet de définir les caractéristiques de la loi de comportement de type élasto-plastique de Von-Mises à écrouissage isotrope linéaire pour un calcul avec Aster.

avmil = **ASTER_VMIS_ISOT_LINE**(
- ♦ **NOM** = nom, [**str**]
- ♦ **ELAS** = elas, [**ELAS**]
- ♦ **ECRO_LINE** = ecro_line, [**ECRO_LINE**]

) # *fin ASTER_VMIS_ISOT_LINE*

NOM : chaîne de caractères définissant le nom de la loi.
ELAS : objet de type **ELAS** définissant la partie linéaire de la loi.
ECRO_LINE : objet de type **ECRO_LINE** définissant la partie non linéaire de la loi.

Partie linéaire. La classe **ELAS** permet de définir les caractéristiques de la partie linéaire.

elas = **ELAS**(
♦ **E** = e, [**float**]
♦ **NU** = nu, [**float**]
♦ **RO** = ro, [**float**]
) # *fin ELAS*

E : réel donnant la valeur du module d'Young.
NU : réel donnant la valeur du coefficient de Poisson.
RO : réel donnant la valeur de la masse volumique.

Partie non linéaire. La classe **ECRO_LINE** permet de définir les caractéristiques de la partie non linéaire.

ecroline = **ECRO_LINE**(
♦ **D_SIGM_EPSI** = d_sigm_epsi, [**float**]
♦ **SY** = sy, [**float**]
) # *fin ECRO_LINE*

D_SIGM_EPSI : réel donnant la valeur de la pente de la courbe de traction.
SY : réel donnant la valeur de la limite d'élasticité.

2.10.21 Classe MATER_RENFORCE

La classe **MATER_RENFORCE** permet de définir les caractéristiques d'un matériau renforcé. On définit les lois de comportement associées d'une part à la matrice du matériau et d'autre part à ses renforcement éventuels.

mat_renf = **MATER_RENFORCE**(
♦ **NOM** = nom, [**str**]
♦ **TYP** = / **'BUHAN_SUDRET'**,
♦ **MATRICE** = matrice, [**COMPORTEMENT**]
◊ **RENFORTS** = renforts, [**list<COMPORTEMENT>**]
) # *fin MATER_RENFORCE*

NOM : chaîne de caractères définissant le nom du matériau renforcé.
TYP : chaîne de caractères définissant le type du modèle de matériau renforcé à choisir parmi :
— **'BUHAN_SUDRET'** : pour le modèle de Buhan et Sudret.

MATRICE : objet de type **COMPORTEMENT** définissant le comportement de la matrice du matériau. Pour l'instant, seul un comportement de type **DRUCK_PRAG_SANS_ECROU** est possible.
RENFORTS : liste d'objets de type **COMPORTEMENT** définissant les comportements des renforcements du matériau. Pour l'instant, seul un comportement de type **ELASTO_PLAST_1D** est possible.

2.10.22 Classe ELASTO_PLAST_1D

La classe **ELASTO_PLAST_1D** permet de définir les caractéristiques d'un matériau élasto-plastique constituant une inclusion 1D.

elpl1d = **ELASTO_PLAST_1D**(
♦ **NOM** = nom, [**str**]
♦ **K** = k, [**float**]
♦ **S** = s, [**float**]
♦ **ETA** = eta, [**float**]
) # *fin ELASTO_PLAST_1D*

NOM : chaîne de caractères définissant le nom du matériau élasto-plastique 1D.
K : réel donnant la valeur du module d'Young du matériau constituant l'inclusion.
S : réel donnant la valeur de la section de l'inclusion.
ETA : réel donnant la valeur du rapport limite en compression / limite en traction du matériau constituant l'inclusion.

2.10.23 Classe COMPOSITE

La classe **COMPOSITE** permet de définir les caractéristiques d'un matériau composite de type multi-fibres ou multi-couches.

compos = **COMPOSITE**(
♦ **NOM** = nom, [**str**]
♦ **TYP** = / **'MULTI_FIBR'**,
♦ **GEOM_FIBRE** = geom_fibre, [**POUT_3D_MULTI**]
/ **'MULTI_COUCH'**,
♦ **GEOM_COUCHE** = geom_couche, [**COQUE_MULTI**]
♦ **AFFECT** = affect, [**list<AFFECT>**]
) # *fin COMPOSITE*

NOM : chaîne de caractères définissant le nom du composite.
TYP : chaîne de caractères définissant le type du modèle de matériau renforcé à choisir parmi :
— **'MULTI_FIBR'** : pour un composite de type multi-fibres,
— **'MULTI_COUCH'** : pour un composite de type multi-couches.
GEOM_FIBRE : objet de type **POUT_3D_MULTI** défini à la section 2.11.15.
GEOM_COUCHE : objet de type **COQUE_MULTI** défini à la section 2.11.16.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de matériau composite. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

```
affect = [ # debut liste
          AFFECT(
          ♦ PROPRIETE = formul, [/ ELAS_LINE_ISO
                                 / VON_MISES_AVEC_ECROU
                                 / ASTER_ELAS
                                 / ASTER_VMIS_ISOT_LINE]
          ♦ LIEU = LIEU(
                   ♦ TYP = / 'GROUP_FIBRE',
                           / 'GROUP_COUCHE',
                   ♦ NOMS = noms, [list<str>]
                   ), # fin LIEU
          ), # fin AFFECT
        ] # fin liste
```

Ainsi, dans le contexte de la classe **COMPOSITE** :

— la propriété doit être un objet **COMPORTEMENT** de type à choisir parmi :
  · **ELAS_LINE_ISO**,
  · **VON_MISES_AVEC_ECROU**,
  · **ASTER_ELAS**,
  · **ASTER_VMIS_ISOT_LINE**.

— la propriété ne peut être appliquée que sur des **'GROUP_FIBRE'** ou des **'GROUP_COUCHE'**.

2.10.24 Classe DIFFUS_LINE_2D

La classe **DIFFUS_LINE_2D** permet de définir les caractéristiques de la loi de comportement linéaire en diffusion 2D (conduction de la chaleur ou écoulement en milieu poreux).

```
diflin2d = DIFFUS_LINE_2D(
           ♦ NOM = nom, [str]
           ♦ AKX = akx, [float]
           ♦ AKY = aky, [float]
           ♦ AKXY = akxy, [float]
           ♦ C = c, [float]
           ) # fin DIFFUS_LINE_2D
```

NOM : chaîne de caractères définissant le nom de la loi.
AKX : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKY : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKXY : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
C : réel donnant la valeur de la capacité calorifique (ou coefficient d'emmagasinement).

2.10.25 Classe DIFFUS_LINE_3D

La classe **DIFFUS_LINE_3D** permet de définir les caractéristiques de la loi de comportement linéaire en diffusion 3D (conduction de la chaleur ou écoulement en milieu poreux).

```
diflin3d = DIFFUS_LINE_3D(
           ♦ NOM = nom, [str]
```

♦ **AKX** = akx, [**float**]
♦ **AKY** = aky, [**float**]
♦ **AKZ** = akz, [**float**]
♦ **AKXY** = akxy, [**float**]
♦ **AKYZ** = akyz, [**float**]
♦ **AKXZ** = akxz, [**float**]
♦ **C** = c, [**float**]
) # *fin DIFFUS_LINE_3D*

NOM : chaîne de caractères définissant le nom de la loi.
AKX : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKY : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKZ : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKXY : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKYZ : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
AKXZ : réel donnant un coefficient du tenseur de conductivité (ou perméabilité).
C : réel donnant la valeur de la capacité calorifique (ou coefficient d'emmagasinement).

2.10.26 Classe ECHANG_LINE

La classe **ECHANG_LINE** permet de définir les caractéristiques de la loi d'échange linéaire en diffusion (conduction de la chaleur ou écoulement en milieu poreux).

echlin = **ECHANG_LINE**(
♦ **NOM** = nom, [**str**]
♦ **ECH** = ech, [**float**]
) # *fin ECHANG_LINE*

NOM : chaîne de caractères définissant le nom de la loi.
ECH : réel donnant la valeur du coefficient d'échange.

2.11 Données relatives aux caractéristiques

2.11.1 Classe CARACTERISTIQUE

La classe **CARACTERISTIQUE** est la classe permettant de définir les caractéristiques géométriques à affecter le cas échéant aux éléments finis utilisés.
Par exemple : caractéristiques de sections et d'inerties pour des éléments de poutre, épaisseurs des éléments de plaques, ...
Elle permet également de définir les caractéristiques des renforts dans un massif 3D ou encore les caractéristiques des poutres multi-fibres ou des coques multi-couches.

cara = **CARACTERISTIQUE**(
♦ **NOM** = nom, [**str**]
♦ **MAIL** = mod, [**MAILLAGE**]
♦ **AFFECT** = affect, [**list<AFFECT>**]
) # *fin CARACTERISTIQUE*

NOM : chaîne de caractères définissant le nom des caractéristiques. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'appliquent les caractéristiques.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de caractéristiques. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

```
affect = [ # debut liste
          AFFECT(
          ♦ PROPRIETE = geom, [GEOMETRIE]
          ♦ LIEU = LIEU(
                    ♦ TYP = / 'GROUP_MAILLE',
                            / 'MAILLE',
                    ♦ NOMS = noms, [list<str>]
                    ), # fin LIEU
          ), # fin AFFECT
        ] # fin liste
```

Ainsi, dans le contexte de la classe **CARACTERISTIQUE** :
- la propriété doit être un objet de type **GEOMETRIE**,
- la propriété ne peut être appliquée que sur des **'MAILLE'** ou des **'GROUP_MAILLE'**.

Néanmoins, la classe **GEOMETRIE** est une classe abstraite et on ne l'instancie pas. On invoque en fait les classes instanciables suivantes qui en dérivent :
- **CONTR_PLANE**,
- **DEFOR_PLANE**,
- **AXISYM**,
- **MASSIF_3D**,
- **BAR_2D**,
- **BAR_3D**,
- **POUT_2D**,
- **POUT_3D**,
- **COQUE**,
- **POUT_3D_MULTI**,
- **COQUE_MULTI**.

2.11.2 Classe CONTR_PLANE

La classe **CONTR_PLANE** permet de définir l'épaisseur de la plaque dans le cadre de l'hypothèse des contraintes planes pour un modèle de mécanique bidimensionnelle (**'MECA_2D_CPLAN'**).

```
cplan = CONTR_PLANE(
        ♦ NOM = nom, [str]
        ♦ EP = ep, [float]
        ) # fin CONTR_PLANE
```

NOM : chaîne de caractères définissant le nom de la caractéristique de contraintes planes.
EP : réel donnant l'épaisseur de la plaque.

2.11.3 Classe DEFOR_PLANE

La classe **DEFOR_PLANE** permet de définir les caractéristiques de renforcement d'un matériau dans l'hypothèse des déformations planes pour un modèle de mécanique bidimensionnelle (**'MECA_2D_DPLAN'**). Il est possible de définir 0, 1 ou 2 renforcements. Chaque renforcement est décrit à l'aide de la classe **RENFOR_DEFOR_PLANE**.

dplan = **DEFOR_PLANE**(
♦ **NOM** = nom, [**str**]
◊ **RENFORTS** = renforts, [**list<RENFOR_DEFOR_PLANE>**]
) # *fin DEFOR_PLANE*

NOM : chaîne de caractères définissant le nom de la caractéristique de déformations planes.
RENFORTS : liste d'objets de type **RENFOR_DEFOR_PLANE** définissant un ensemble des renforcements.

Renforcement en déformations planes. La classe **RENFOR_DEFOR_PLANE** permet de définir une ou plusieurs phases de renforcement d'un matériau dans l'hypothèse des déformations planes.

renf_dplan = **RENFOR_DEFOR_PLANE**(
♦ **TYP** = / **'HOMOGEN'**,
♦ **X1** = x1, [**float**]
♦ **FV1** = fv1, [**float**]
/ **'RADIAL'**,
♦ **X2** = x2, [**float**]
♦ **Y2** = y2, [**float**]
♦ **FV2** = fv2, [**float**]
♦ **SA** = sa, [**float**]
) # *fin RENFOR_DEFOR_PLANE*

TYP : chaîne de caractères définissant le type de la phase de renforcement à choisir parmi :
— **'HOMOGEN'** : pour un renforcement homogène,
— **'RADIAL'** : pour un renforcement radial.
X1 : réel donnant l'angle entre la direction des inclusions de la phase et l'axe horizontal.
FV1 : réel donnant la fraction volumique des inclusions de la phase.
X2 : réel donnant la coordonnée x du point de convergence des inclusions de la phase.
Y2 : réel donnant la coordonnée y du point de convergence des inclusions de la phase.
FV2 : réel donnant la fraction volumique des inclusions de la phase à une distance unité du point de convergence des inclusions.
SA : réel donnant la section d'une inclusion de la phase.

2.11.4 Classe AXISYM

La classe **AXISYM** permet de définir les caractéristiques de renforcement d'un matériau dans l'hypothèse d'axisymétrie pour un modèle de mécanique bidimensionnelle (**'AXISYM'**). Il est possible de définir 0, 1 ou 2 renforcements. Chaque renforcement est décrit à l'aide de la classe **RENFOR_AXISYM**.

```
axi = AXISYM(
      ♦ NOM = nom, [str]
      ◊ RENFORTS = renforts, [list<RENFOR_AXISYM>]
      ) # fin AXISYM
```

NOM : chaîne de caractères définissant le nom de la caractéristique d'axisymétrie.
RENFORTS : liste d'objets de type **RENFOR_AXISYM** définissant un ensemble des renforcements.

Renforcement en axisymétrie. La classe **RENFOR_AXISYM** permet de définir une ou plusieurs phases de renforcement d'un matériau dans l'hypothèse d'axisymétrie.

```
renf_axi = RENFOR_AXISYM(
           ♦ TYP = / 'HOMOGEN',
                     ♦ FV1 = fv1, [float]
                   / 'RADIAL',
                     ♦ FV2 = fv2, [float]
                   / 'DIVERG_CYLINDR',
                     ♦ X3 = x3, [float]
                     ♦ FV3 = fv3, [float]
                   / 'DIVERG_SPHER',
                     ♦ X4 = x4, [float]
                     ♦ FV4 = fv4, [float]
           ♦ SA = sa, [float]
           ) # fin RENFOR_AXISYM
```

TYP : chaîne de caractères définissant le type de la phase de renforcement à choisir parmi :
— **'HOMOGEN'** : pour un renforcement homogène,
— **'RADIAL'** : pour un renforcement radial.
— **'DIVERG_CYLINDR'** : pour un renforcement divergent cylindrique.
— **'DIVERG_SPHER'** : pour un renforcement divergent sphérique.
FV1 : réel donnant la fraction volumique des inclusions de la phase.
FV2 : réel donnant la fraction volumique des inclusions de la phase à une distance unité de l'axe de symétrie.
X3 : réel donnant l'angle entre la direction des inclusions de la phase et l'axe de symétrie.
FV3 : réel donnant la fraction volumique des inclusions de la phase à une distance unité de l'axe de symétrie.
X4 : réel donnant la coordonnée z du point de convergence des inclusions de la phase.
FV4 : réel donnant la fraction volumique des inclusions de la phase à une distance unité du point de convergence des inclusions de la phase.
SA : réel donnant la section d'une inclusion de la phase.

2.11.5 Classe MASSIF_3D

La classe **MASSIF_3D** permet de définir les caractéristiques de renforcement d'un matériau pour un modèle de mécanique tridimensionnelle (**'MECA_3D'**). Il est possible de définir 0, 1 ou 2 renforcements. Chaque renforcement est décrit à l'aide de la classe **RENFOR_3D**.

```
mas3d = MASSIF_3D(
        ♦ NOM = nom, [str]
        ◊ RENFORTS = renforts, [list<RENFOR_3D>]
        ) # fin MASSIF_3D
```

NOM : chaîne de caractères définissant le nom de la caractéristique de massif 3D.
RENFORTS : liste d'objets de type **RENFOR_3D** définissant un ensemble des renforcements.
La classe **RENFOR_3D** permet de définir une ou plusieurs phases de renforcement d'un matériau pour un modèle de mécanique tridimensionnelle.

```
renf3d = RENFOR_3D(
        ♦ TYP = / 'HOMOGEN',
                  ♦ A1 = a1, [float]
                  ♦ B1 = b1, [float]
                  ♦ FV1 = fv1, [float]
                / 'RADIAL',
                  ♦ X2 = x2, [float]
                  ♦ Y2 = y2, [float]
                  ♦ Z2 = z2, [float]
                  ♦ A2 = a2, [float]
                  ♦ B2 = b2, [float]
                  ♦ FV2 = fv2, [float]
                / 'DIVERG_CYLINDR',
                  ♦ X3 = x3, [float]
                  ♦ Y3 = y3, [float]
                  ♦ Z3 = z3, [float]
                  ♦ A3 = a3, [float]
                  ♦ B3 = b3, [float]
                  ♦ C3 = c3, [float]
                  ♦ FV3 = fv3, [float]
                / 'DIVERG_SPHER',
                  ♦ X4 = x4, [float]
                  ♦ Y4 = y4, [float]
                  ♦ Z4 = z4, [float]
                  ♦ FV4 = fv4, [float]
        ♦ SA = sa, [float]
        ) # fin RENFOR_3D
```

TYP : chaîne de caractères définissant le type de la phase de renforcement à choisir parmi :
— **'HOMOGEN'** : pour un renforcement homogène,
— **'RADIAL'** : pour un renforcement radial.
— **'DIVERG_CYLINDR'** : pour un renforcement divergent cylindrique.
— **'DIVERG_SPHER'** : pour un renforcement divergent sphérique.
A1 : réel donnant l'angle entre la projection de la direction de renforcement sur le plan $z = 0$ et l'axe des x.
B1 : réel donnant l'angle entre la direction de renforcement et l'horizontale.
FV1 : réel donnant la fraction volumique des inclusions de la phase.

X2 : réel donnant la coordonnée x d'un point de l'axe de symétrie.
Y2 : réel donnant la coordonnée y d'un point de l'axe de symétrie.
Z2 : réel donnant la coordonnée z d'un point de l'axe de symétrie.
A2 : réel donnant l'angle entre la projection de l'axe de symétrie sur le plan $z = 0$ et l'axe des x.
B2 : réel donnant l'angle entre la direction de l'axe de symétrie et l'horizontale.
FV2 : réel donnant la fraction volumique des inclusions de la phase à une distance unité de l'axe de symétrie.
X3 : réel donnant la coordonnée x d'un point de l'axe de symétrie.
Y3 : réel donnant la coordonnée y d'un point de l'axe de symétrie.
Z3 : réel donnant la coordonnée z d'un point de l'axe de symétrie.
A3 : réel donnant l'angle entre la projection de l'axe de symétrie sur le plan $z = 0$ et l'axe des x.
B3 : réel donnant l'angle entre la direction de l'axe de symétrie et l'horizontale.
C3 : réel donnant l'angle entre la direction de renforcement et l'axe de symétrie dans un plan méridien.
FV3 : réel donnant la fraction volumique des inclusions de la phase à une distance unité de l'axe de symétrie.
X4 : réel donnant la coordonnée x du point de convergence des inclusions.
Y4 : réel donnant la coordonnée y du point de convergence des inclusions.
Z4 : réel donnant la coordonnée z du point de convergence des inclusions.
FV4 : réel donnant la fraction volumique des inclusions de la phase à une distance unité du point de convergence des inclusions.
SA : réel donnant la section d'une inclusion de la phase.

2.11.6 Classe BAR_2D

La classe **BAR_2D** permet de définir les caractéristiques de barre pour un modèle de barre bidimensionnelle (**'BAR_2D'**).

```
bar2d = BAR_2D(
        ♦ NOM = nom, [str]
        ♦ S = s [float]
        ◊ INTER = inter, [/ BAR_FROT_LINE
                         / BAR_FROT_PLAS
                         / BAR_FROT_BILINE
                         / BAR_FROT_RACINE]
        ) # fin BAR_2D
```

NOM : chaîne de caractères définissant le nom de la caractéristique de barre bidimensionnelle.
S : réel donnant la section de la barre,
INTER : objet de type **BAR_FROT_LINE**, **BAR_FROT_PLAS**, **BAR_FROT_BILINE** ou **BAR_FROT_RACINE** décrivant l'intéraction entre la barre et le milieu qui l'entoure. Il doit être renseigné dans le cas d'une formulation mécanique de type **'BAR_2D_FROT'**.

2.11.7 Classe BAR_3D

La classe **BAR_3D** permet de définir les caractéristiques de barre pour un modèle de barre tridimensionnelle (**'BAR_3D'**).

bar3d = **BAR_3D**(
♦ **NOM** = nom, [**str**]
♦ **S** = s [**float**]
◊ **INTER** = inter, [/ **BAR_FROT_LINE**
/ **BAR_FROT_PLAS**
/ **BAR_FROT_BILINE**
/ **BAR_FROT_RACINE**]
) # *fin BAR_3D*

NOM : chaîne de caractères définissant le nom de la caractéristique de barre tridimensionnelle.
S : réel donnant la section de la barre,
INTER : objet de type **BAR_FROT_LINE**, **BAR_FROT_PLAS**, **BAR_FROT_BILINE** ou **BAR_FROT_RACINE** décrivant l'intéraction entre la barre et le milieu qui l'entoure. Il doit être renseigné dans le cas d'une formulation mécanique de type **'BAR_3D_FROT'**.

2.11.8 Classe BAR_FROT_LINE

La classe **BAR_FROT_LINE** permet de définir les caractéristiques d'une intéraction de type élasticité linéaire entre une barre (bidimensionnelle ou tridimensionnelle) et le milieu qui l'entoure.

line = **BAR_FROT_LINE**(
♦ **COEF_INT** = coef_int [**float**]
) # *fin BAR_FROT_LINE*

COEF_INT : réel donnant la valeur du coefficient d'intéraction.

2.11.9 Classe BAR_FROT_PLAS

La classe **BAR_FROT_PLAS** permet de définir les caractéristiques d'une intéraction de type élasticité linéaire et plasticité parfaite entre une barre (bidimensionnelle ou tridimensionnelle) et le milieu qui l'entoure.

plas = **BAR_FROT_PLAS**(
♦ **COEF_INT** = coef_int [**float**]
♦ **FORC_MAX** = forc_max [**float**]
) # *fin BAR_FROT_PLAS*

COEF_INT : réel donnant la valeur du coefficient d'intéraction,
FORC_MAX : réel donnant la valeur maximale de la force linéique d'intéraction.

2.11.10 Classe BAR_FROT_BILINE

La classe **BAR_FROT_BILINE** permet de définir les caractéristiques d'une intéraction de type élasticité bilinéaire et plasticité parfaite entre une barre (bidimensionnelle ou tridimensionnelle) et le milieu qui l'entoure.

biline = **BAR_FROT_BILINE**(
- **COEF_INIT** = coef_init [**float**]
- **COEF_SEC** = coef_sec [**float**]
- **FORC_SEUIL** = forc_seuil [**float**]
- **FORC_MAX** = forc_max [**float**]

) # *fin BAR_FROT_BILINE*

COEF_INIT : réel donnant la valeur du coefficient d'intéraction initial,
COEF_SEC : réel donnant la valeur du coefficient d'intéraction secondaire,
FORC_SEUIL : réel donnant la valeur seuil de la force linéique d'intéraction,
FORC_MAX : réel donnant la valeur maximale de la force linéique d'intéraction.

2.11.11 Classe BAR_FROT_RACINE

La classe **BAR_FROT_RACINE** permet de définir les caractéristiques d'une intéraction de type élasticité loi racine et plasticité parfaite entre une barre (bidimensionnelle ou tridimensionnelle) et le milieu qui l'entoure.

racine = **BAR_FROT_RACINE**(
- **COEF_INIT** = coef_init [**float**]
- **DEPLA_REL** = depla_rel [**float**]
- **FORC_MAX** = forc_max [**float**]

) # *fin BAR_FROT_RACINE*

COEF_INIT : réel donnant la valeur du coefficient d'intéraction initial,
DEPLA_REL : réel donnant la valeur du déplacement relatif pour lequel la force d'intéraction atteint la valeur maximale,
FORC_MAX : réel donnant la valeur maximale de la force linéique d'intéraction.

2.11.12 Classe POUT_2D

La classe **POUT_2D** permet de définir les caractéristiques de poutre pour un modèle de poutre bidimensionnelle (**'POUT_2D'**).

pout2d = **POUT_2D**(
- **NOM** = nom, [**str**]
- **S** = s [**float**]
- **SR** = sr [**float**]
- **VIN** = vin [**float**]
- **YG** = yg [**float**]

) # *fin POUT_2D*

NOM : chaîne de caractères définissant le nom de la caractéristique de poutre bidimensionnelle.
S : réel donnant la section droite.
SR : réel donnant la section réduite au cisaillement.
VIN : réel donnant la valeur de l'inertie principale
YG : réel donnant l'ordonnée de l'axe des centres de gravité des sections.

2.11.13 Classe POUT_3D

La classe **POUT_3D** permet de définir les caractéristiques de poutre pour un modèle de poutre tridimensionnelle **'POUT_3D'**.

pout3d = **POUT_3D**(
- ♦ **NOM** = nom, [**str**]
- ♦ **S** = s [**float**]
- ♦ **S2** = s2 [**float**]
- ♦ **S3** = s3 [**float**]
- ♦ **VI1** = vi1 [**float**]
- ♦ **VI2** = vi2 [**float**]
- ♦ **VI3** = vi3 [**float**]
- ♦ **YG** = yg [**float**]
- ♦ **ZG** = zg [**float**]
- ♦ **YC** = yc [**float**]
- ♦ **ZC** = zc [**float**]
- ♦ **V** = v, [**list<float>**]

) # *fin POUT_3D*

NOM : chaîne de caractères définissant le nom de la caractéristique de poutre tridimensionnelle.
S : réel donnant la section droite.
S2 : réel donnant la section réduite au cisaillement dans la direction 2.
S3 : réel donnant la section réduite au cisaillement dans la direction 3.
VI1 : réel donnant la valeur de l'inertie de torsion.
VI2 : réel donnant la valeur de l'inertie principale par rapport à l'axe 2.
VI3 : réel donnant la valeur de l'inertie principale par rapport à l'axe 3.
YG : réel donnant la coordonnée dans l'axe 2 de l'axe des centres de gravité des sections.
ZG : réel donnant la coordonnée dans l'axe 3 de l'axe des centres de gravité des sections.
YC : réel donnant la coordonnée dans l'axe 2 de l'axe des centres de torsion des sections.
ZC : réel donnant la coordonnée dans l'axe 3 de l'axe des centres de torsion des sections.
V : liste de 3 réels donnant les coordonnées de l'axe 2.

2.11.14 Classe COQUE

La classe **COQUE** permet de définir les caractéristiques de coque standard pour un modèle de coque mince (**'COQ_MINC'**) ou épaisse (**'COQ_EPAIS'**) .

coq = **COQUE**(
- ♦ **NOM** = nom, [**str**]
- ♦ **EP** = eps [**float**]

) # *fin COQUE*

NOM : chaîne de caractères définissant le nom de la caractéristique de coque.
EP : réel donnant l'épaisseur de la coque.

2.11.15 Classe POUT_3D_MULTI

La classe **POUT_3D_MULTI** permet de définir les caractéristiques d'une poutre multi-fibres pour un modèle de poutre épaisse tridimensionnelle (**'POUT_3D'**). On constitue des groupes de fibres via la classe **GROUP_FIBRE** et on définit les caractéristiques des fibres via la classe **FIBRE**.

pout_mul = **POUT_3D_MULTI**(
♦ **NOM** = nom, [**str**]
♦ **YC** = yc [**float**]
♦ **ZC** = zc [**float**]
◊ **VITORS** = vitors [**float**]
♦ **V** = v, [**list<float>**]
♦ **GRPES** = grpes, [**list<GROUP_FIBRE>**]
) # *fin POUT_3D_MULTI*

NOM : chaîne de caractères définissant le nom de la caractéristique de poutre multi-fibres.
YC : réel donnant la coordonnée dans l'axe 2 de l'axe des centres de torsion des sections.
ZC : réel donnant la coordonnée dans l'axe 3 de l'axe des centres de torsion des sections.
VITORS : réel donnant l'inertie de torsion (= 0 par défaut).
V : liste de réels donnant les coordonnées de l'axe 2.
GRPES : liste d'objets de type **GROUP_FIBRE** définissant les groupes de fibres.

Groupes de fibres. La classe **GROUP_FIBRE** permet de regrouper un ensemble de fibres et de l'identifier par un nom (par exemple, le groupe des fibres constituant la partie béton d'une section de poutre en béton armé).

gr_fibr = **GROUP_FIBRE**(
♦ **NOM** = nom, [**str**]
♦ **FIBRES** = fibres, [**list<FIBRE>**]
) # *fin GROUP_FIBRE*

NOM : chaîne de caractères définissant le nom du groupe de fibres.
FIBRES : liste d'objets de type **FIBRE** définissant les caractéristiques géométriques des fibres du groupe.

Fibres. La classe **FIBRE** permet de définir les caractéristiques géométriques d'une fibre (surface et positionnement dans la section).

fibr = **FIBRE**(
♦ **SF** = sf, [**str**]
♦ **X2** = x2 [**float**]
♦ **X3** = x3 [**float**]
) # *fin FIBRE*

SF : réel donnant l'aire de la section de la fibre.
X2 : réel donnant la coordonnée dans l'axe 2 du centre de la fibre.
X3 : réel donnant la coordonnée dans l'axe 3 du centre de la fibre.

2.11.16 Classe COQUE_MULTI

La classe **COQUE_MULTI** permet de définir les caractéristiques d'une coque multi-couches pour un modèle de coque mince (**'COQ_MINC'**) ou épaisse (**'COQ_EPAIS'**). On constitue des groupes de couches via la classe **GROUP_COUCHE** et on définit les caractéristiques des couches via la classe **COUCHE**.

coq_mult = **COQUE_MULTI**(
♦ **NOM** = nom, [**str**]
♦ **GRPES** = grpes, [**list<GROUP_COUCHE>**]
) # *fin COQUE_MULTI*

NOM : chaîne de caractères définissant le nom de la caractéristique de coque multi-couches.
GRPES : liste d'objets de type **GROUP_COUCHE** définissant les groupes de couches.

Groupes de couches. La classe **GROUP_COUCHE** permet de regrouper un ensemble de couches et de l'identifier par un nom.

gr_couch = **GROUP_COUCHE**(
♦ **NOM** = nom, [**str**]
♦ **COUCHES** = couches, [**list<COUCHE>**]
) # *fin GROUP_COUCHE*

NOM : chaîne de caractères définissant le nom du groupe de couches.
COUCHES : liste d'objets de type **COUCHE** définissant les caractéristiques géométriques des couches du groupe.

Couches. La classe **COUCHE** permet de définir les caractéristiques géométriques d'une couche (épaisseur et excentrement).

couch = **COUCHE**(
♦ **EP** = ep, [**str**]
♦ **EXC** = exc [**float**]
) # *fin COUCHE*

EP : réel donnant l'épaisseur de la couche.
EXC : réel donnant l'excentricité de la couche.

2.12 Données relatives aux activations

2.12.1 Classe ACTIVATION

La classe **ACTIVATION** est la classe permettant de définir les groupes d'éléments du maillage à activer.

activ = **ACTIVATION**(
♦ **NOM** = nom, [**str**]
♦ **MAIL** = mail, [**MAILLAGE**]
♦ **GRPES** = grpes, [**list<GROUPE>**]
) # *fin ACTIVATION*

NOM : chaîne de caractères définissant le nom de l'activation.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'applique l'activation.
AFFECT : liste d'objets de type **GROUPE** définissant les groupes d'éléments du maillage qui sont activés.

2.13 Données relatives aux liaisons

2.13.1 Classe LIAISON

La classe **LIAISON** est la classe permettant de définir des liaisons entre parties du maillage. Il peut s'agir d'introduire des relations linéaires entre degrés de liberté, des matrices (rigidité, masse, amortissement) élémentaires, ou encore des conditions de contact ou de joint entre zones.

liais = **LIAISON**(
♦ **NOM** = nom, [**str**]
♦ **MAIL** = mail, [**MAILLAGE**]
♦ **AFFECT** = affect, [**list<AFFECT>**]
) # *fin LIAISON*

NOM : chaîne de caractères définissant le nom de la liaison.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'applique la liaison.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de la liaison. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

affect = [# *debut liste*
AFFECT(
♦ **PROPRIETE** = ppte, [/ **RELA_LINE**
/ **MATR_ELEM**
/ **CONTACT**
/ **JOINT**]
♦ **LIEU** = **LIEU**(
♦ # *si **RELA_LINE***
/ **TYP** = / **'NOEUD'**,
*si **MATR_ELEM***
/ **TYP** = / **'NOEUD'**,
*si **CONTACT***
/ **TYP** = / **'GROUP_ARETE'**,
/ **'GROUP_FACE'**,
*si **JOINT***
/ **TYP** = / **'GROUP_ARETE'**,
/ **'GROUP_FACE'**,
♦ **NOMS** = noms, [**list<str>**]

```
                ), # fin LIEU
            ), # fin AFFECT
        ] # fin liste
```

Ainsi, dans le contexte de la classe **LIAISON** :
- la propriété doit être un objet de type **RELA_LINE**, **MATR_ELEM**, **CONTACT** ou **JOINT**,
- suivant le type de propriété, celle-ci ne peut être appliquée que sur des **'NOEUD'**, des **'GROUP_ARETE'** ou des **'GROUP_FACE'**.

2.13.2 Classe RELA_LINE

La classe **RELA_LINE** permet de définir une relation linéaire entre degrés de liberté de la forme :

$$\sum_i \alpha_i u_i = \beta$$

où les u_i sont les degrés de liberté, α_i les coefficients de la relation et où β est la constante de la relation actuellement imposée à la valeur 0.

```
rela_line = RELA_LINE(
            ♦ NOM = nom, [str]
            ♦ COEF = coef, [list<float>]
            ♦ CSTE = cste, [float]
            ♦ DDL = ddl, [list <| 'U'
                               | 'V'
                               | 'W'
                               | 'RX'
                               | 'RY'
                               | 'RZ'
                              >] # fin list
            ) # fin RELA_LINE
```

NOM : chaîne de caractères définissant le nom de la relation linéaire.
COEF : liste de réels donnant les coefficients α_i de la relation linéaire.
CSTE : réel donnant le terme constant β de la relation linéaire. Actuellement, cette constante doit être fixée à 0.
DDL : liste de chaînes de caractères à choisir parmi **'U'**, **'V'**, **'W'**, **'RX'**, **'RY'**, **'RZ'** indiquant les degrés de liberté impliqués dans la relation linéaire. Elle doit avoir la même taille que la liste **COEF**.

La donnée **NOMS** de l'**AFFECT** correspondant contient les noms des noeuds concernés par la relation linéaire. Cette liste doit avoir une taille égale à celle de la donnée **DDL** ou bien à un multiple de celle-ci, ce qui permet d'appliquer la même relation linéaire sur plusieurs ensembles de noeuds.
Par exemple, avec le codage suivant :

```
liais = LIAISON(NOM = 'liais',
                MAIL = mail,
                AFFECT = [AFFECT(PROPRIETE = RELA_LINE(NOM = 'rela',
                                                       COEF = [1., -1.],
                                                       CSTE = 0.,
                                                       DDL = ['V','V']),
                                 LIEU = LIEU(NOMS = ['2','3', '84','93'],
                                             TYP = 'NOEUD'))])
```

la relation linéaire $1.u_1 - 1.u_2 = 0$ sera appliquée :
- d'une part aux noeuds 2 et 3 : $1.V(2) - 1.V(3) = 0$,
- d'autre part aux noeuds 84 et 93 : $1.V(84) - 1.V(93) = 0$

où $V(i)$ signifie valeur du degré de liberté **'V'** au noeud i.

2.13.3 Classe MATR_ELEM

La classe **MATR_ELEM** permet de définir une matrice élémentaire de rigidité, de masse ou d'amortissement.

matr_elem = **MATR_ELEM**(
- ♦ **NOM** = nom, [**str**]
- ♦ **MATRICE** = matr, [**list<list<float>>**]
- ♦ **TYP** = / **'RIGI'**,
  / **'MASSE'**,
  / **'AMOR'**,
- ♦ **NB_NOEUD** = nb_noeud, [**int**]
- ♦ **NB_DDL_NOEUD** = nb_ddl_noeud, [**int**]

) # *fin MATR_ELEM*

NOM : chaîne de caractères définissant le nom de la matrice élémentaire.
MATRICE : liste de listes de réels donnant les coefficients de la matrice. Cette liste est de taille **NB_NOEUD** x **NB_DDL_NOEUD** et chaque sous-liste est également de cette taille, de sorte que la matrice décrite est carrée d'ordre **NB_NOEUD** x **NB_DDL_NOEUD**.
TYP : chaîne de caractères indiquant si la matrice élémentaire est une matrice de rigidité (**'RIGI'**), de masse (**'MASSE'**) ou d'amortissement (**'AMOR'**).
NB_NOEUD : entier donnant le nombre de noeuds de l'élément (virtuel) associé à la matrice.
NB_DDL_NOEUD : entier donnant le nombre de degrés de liberté portés par chaque noeud de l'élément (virtuel) associé à la matrice.

Le **LIEU** d'affectation de la matrice élémentaire peut être de type **'NOEUD'** ou **'GROUP_NOEUD'**. Il doit permettre de définir une liste de noeuds cohérente avec la donnée **NB_NOEUD** de **MATR_ELEM**. Ainsi, cette liste doit avoir une taille égale à un multiple de **NB_NOEUD**, ce qui permet d'appliquer la même matrice élémentaire sur plusieurs ensembles de noeuds.
Par exemple, avec le codage suivant :

```
liais = LIAISON(NOM = 'liais',
                MAIL = mail,
                AFFECT = [AFFECT(PROPRIETE = MATR_ELEM(NOM = 'matr',
                                                       MATRICE = [[ k , 0., 0., -k , 0., 0.],
                                                                  [ 0., 0., 0., 0., 0., 0.],
                                                                  [ 0., 0., 0., 0., 0., 0.],
                                                                  [-k , 0., 0., k , 0., 0.],
                                                                  [ 0., 0., 0., 0., 0., 0.],
                                                                  [ 0., 0., 0., 0., 0., 0.]],
                                                       TYP = 'RIGI',
                                                       NB_NOEUD = 2,
                                                       NB_DDL_NOEUD = 3),
                                 LIEU = LIEU(NOMS = ['84','93', '85','94'],
                                             TYP = 'NOEUD'))])
```

une matrice de rigidité élémentaire sera prise en compte :
- d'une part pour l'ensemble des noeuds 84, 93,
- d'autre part pour l'ensemble des noeuds 85, 94.

Chacun de ces ensembles de 2 noeuds portant 3 degrés de liberté, la matrice est d'ordre $2.3 = 6$. L'introduction de cette matrice élémentaire conduit ainsi à ajouter les deux contributions

$$\frac{1}{2} < u_1(84), u_1(93) > \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_1(84) \\ u_1(93) \end{Bmatrix} = \frac{1}{2} k [u_1(84) - u_1(93)]^2$$

et

$$\frac{1}{2} < u_1(85), u_1(94) > \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_1(85) \\ u_1(94) \end{Bmatrix} = \frac{1}{2} k [u_1(85) - u_1(94)]^2$$

à l'énergie du système, où $u_1(i)$ signifie valeur du premier degré de liberté au noeud i.
En particulier, si l'on choisit une grande valeur pour k (par exemple $1.e12$), cette matrice élémentaire conduit donc à imposer par pénalisation, les 2 relations linéaires $u_1(84) = u_1(93)$ et $u_1(85) = u_1(94)$.

2.13.4 Classe CONTACT

La classe **CONTACT** permet de définir un contact entre 2 zones au niveau de leur frontière surfacique (en 3D) ou linéique (en 2D).

contact = **CONTACT**(
♦ **NOM** = nom, [**str**]
♦ **CONTACT_INI** = / **'OUI'**,
/ **'NON'**,
♦ **COMPORT** = comport, [/ **ADHER**
/ **FROT_COUL**
/ **GLISS**]
◊ **EPAIS_CPLAN** = epais, [**float**]
) # *fin CONTACT*

NOM : chaîne de caractères définissant le nom du contact.
CONTACT_INI : chaîne de caractères indiquant si les éléments sont initialement en contact (**'OUI'**) ou pas (**'NON'**).
COMPORT : objet de type **ADHER**, **FROT_COUL** ou **GLISS** donnant les caractéristique du contact considéré.
EPAIS_CPLAN : en contrainte plane, réel donnant l'épaisseur de la structure.

La donnée **NOMS** de l'**AFFECT** correspondant doit être constituée d'une liste de 2 chaînes de caractères donnant les noms des 2 groupes (**'GROUP_ARETE'** ou **'GROUP_FACE'**) en vis à vis lors du contact. Les 2 zones en contact ne doivent pas coïncider géométriquement de façon parfaite : lors de la construction du maillage, il convient d'introduire un écart infinitésimal entre les 2 zones. Cette limitation facilite le test de l'orientation des éléments de contact créés de façon sous-jacente pour le solveur CESAR.

Adhérence. La classe **ADHER** permet de définir une loi de comportement de type adhérence dans une zone de contact.

adher = **ADHER**(
♦ **COEF_RIGI** = coef, [**float**]
) # *fin ADHER*

COEF_RIGI : réel donnant la valeur du coefficient de rigidité du matériau fictif de contact.

Frottement de Coulomb. La classe **FROT_COUL** permet de définir une loi de comportement de type frottement de Coulomb dans une zone de contact.

frot = **FROT_COUL**(
♦ **COEF_RIGI** = coef, [**float**]
♦ **TRACT** = tract, [**float**]
♦ **COHES** = cohes, [**float**]
♦ **ANG_FROT** = frot, [**float**]
♦ **ANG_DILAT** = dilat, [**float**]
) # *fin FROT_COUL*

COEF_RIGI : réel donnant la valeur du coefficient de rigidité du matériau fictif de contact.
TRACT : réel donnant la valeur de la résistance à la traction.
COHES : réel donnant la valeur de la cohésion.
ANG_FROT : réel donnant la valeur en degré de l'angle de frottement.
ANG_DILAT : réel donnant la valeur en degré de l'angle de dilatance.

Glissement. La classe **GLISS** permet de définir une loi de comportement de type glissement dans une zone de contact.

gliss = **GLISS**(
♦ **COEF_RIGI** = coef, [**float**]
♦ **TRACT** = tract, [**float**]
) # *fin GLISS*

COEF_RIGI : réel donnant la valeur du coefficient de rigidité du matériau fictif de contact.
TRACT : réel donnant la valeur de la résistance à la traction.

2.13.5 Classe JOINT

La classe **JOINT** permet de définir un joint entre 2 zones au niveau de leur frontière surfacique (en 3D) ou linéique (en 2D).
La mise en données est analogue à celle de la classe **CONTACT** et utilise les mêmes classes **ADHER**, **FROT_COUL** et **GLISS**. Néanmoins, ce modèle n'est pas utilisable en contrainte plane (d'où l'absence du mot-clé **EPAIS_CPLAN**).

joint = **JOINT**(
♦ **NOM** = nom, [**str**]
♦ **CONTACT_INI** = / **'OUI'**,
/ **'NON'**,
♦ **COMPORT** = comport, [/ **ADHER**
/ **FROT_COUL**
/ **GLISS**]
) # *fin JOINT*

2.14 Données relatives aux conditions limites

2.14.1 Classe COND_LIMITE

La classe **COND_LIMITE** est la classe permettant de définir les conditions aux limites à affecter au modèle.

```
cond = COND_LIMITE(
        ♦ NOM = nom, [str]
        ♦ MAIL = mod, [MAILLAGE]
        ♦ AFFECT = affect, [list<AFFECT>]
        ) # fin COND_LIMITE
```

NOM : chaîne de caractères définissant le nom des conditions limites. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'appliquent les conditions limites.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de conditions limites. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

```
affect = [ # debut liste
          AFFECT(
          ♦ PROPRIETE = cond, [CONDITION]
          ♦ LIEU = LIEU(
                    ♦ TYP = / 'GROUP_FACE',
                            / 'FACE',
                            / 'GROUP_ARETE',
                            / 'ARETE',
                            / 'GROUP_NOEUD',
                            / 'NOEUD',
                    ♦ NOMS = noms, [list<str>]
                    ), # fin LIEU
          ), # fin AFFECT
        ] # fin liste
```

Ainsi, dans le contexte de la classe **COND_LIMITE** :

- la propriété doit être un objet de type **CONDITION**,
- la propriété ne peut être appliquée que sur des entités de type :
  - **'FACE'** ou **'GROUP_FACE'**,
  - **'ARETE'** ou **'GROUP_ARETE'**,
  - **'NOEUD'** ou **'GROUP_NOEUD'**.

Néanmoins, la classe **CONDITION** est une classe abstraite et on ne l'instancie pas. On invoque en fait les classes instanciables suivantes qui en dérivent :

- **DDL_IMPOSE**.

2.14.2 Classe DDL_IMPOSE

La classe **DDL_IMPOSE** permet de définir des valeurs imposées de degrés de liberté.

ddl = **DDL_IMPOSE**(
♦ | **U** = u, [**str**]
| **V** = v, [**str**]
| **W** = w, [**str**]
| **RX** = rx, [**str**]
| **RY** = ry, [**str**]
| **RZ** = rz, [**str**]
| **DREL** = drel, [**str**]
◊ **BASE** = base, [**BASE_2D**, **BASE_3D**]
) # *fin DDL_IMPOSE*

U : déplacement u dans la base du maillage ou celle spécifiée via **BASE**.
V : déplacement v dans la base du maillage ou celle spécifiée via **BASE**.
W : déplacement w dans la base du maillage ou celle spécifiée via **BASE**.
RX : rotation r_x dans la base du maillage ou celle spécifiée via **BASE**.
RY : rotation r_y dans la base du maillage ou celle spécifiée via **BASE**.
RZ : rotation r_z dans la base du maillage ou celle spécifiée via **BASE**.
DREL : déplacement relatif *drel* dans l'axe d'une barre frottante.
BASE : objet de type **BASE_2D** (resp. **BASE_3D**) définissant une base en dimension 2 (resp. 3).

Changement de base. La classe **BASE_2D** (resp. **BASE_3D**) permet de définir un changement de base par rapport à la base d'un maillage 2D (resp. 3D).

base2d = **BASE_2D**(
♦ / **VECT** = vect, [**list<float>**]
/ **ANGL** = angl, [**float**]
) # *fin BASE_2D*

VECT : liste de 2 réels donnant les composantes dans la base du maillage du premier vecteur de la base.
ANGL : réel donnant la valeur en degrés de l'angle définissant le premier vecteur de la base.

base3d = **BASE_3D**(
♦ / **VECT** = vect, [**list<float>**]
/ **ANGL** = angl, [**float**]
) # *fin BASE_3D*

VECT : liste de 6 réels donnant les composantes dans la base du maillage des 2 premiers vecteurs de la base.
ANGL : liste de 3 réels donnant les valeurs en degrés des 3 angles nautiques définissant la base.

2.15 Données relatives aux chargements

2.15.1 Classe CAS_CHARGEMENT

La classe **CAS_CHARGEMENT** est la classe permettant de définir les cas de chargements à affecter au modèle. On constitue une combinaisons linéaire de chargements : chaque chargement est défini via la classe **CHARGEMENT** et affecté d'un coefficient multiplicateur.

cas = **CAS_CHARGEMENT**(
♦ **NOM** = nom, [**str**]
♦ **CHARG** = char, [**list<CHARGEMENT>**]
♦ **COEF** = coef, [**list<float>**]
) # *fin CAS_CHARGEMENT*

NOM : chaîne de caractères définissant le nom du cas de chargement.
CHARG : liste d'objets de type **CHARGEMENT** définissant les chargements de la combinaison linéaire.
COEF : liste de réels définissant les coefficients multiplicateurs des chargements de la combinaison linéaire. Les listes **COEF** et **CHARG** doivent avoir le même cardinal.

2.15.2 Classe CHARGEMENT

La classe **CHARGEMENT** est la classe permettant de définir un chargement intervenant dans un cas de chargement.

char = **CHARGEMENT**(
♦ **NOM** = nom, [**str**]
♦ **MAIL** = mod, [**MAILLAGE**]
♦ **AFFECT** = affect, [**list<AFFECT>**]
) # *fin CHARGEMENT*

NOM : chaîne de caractères définissant le nom du chargement. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
MAIL : objet de type **MAILLAGE** définissant le maillage auquel s'applique le chargement.
AFFECT : liste d'objets de type **AFFECT** définissant les affectations de chargements. On précise ci-dessous l'utilisation autorisée de la classe **AFFECT**.

affect = [# *debut liste*
AFFECT(
♦ **PROPRIETE** = charg, [**CHARGE**]
♦ **LIEU** = **LIEU**(
♦ **TYP** = / # *pour une CHARGE_ELEMENT*
/ **'GROUP_MAILLE'**,
/ **'MAILLE'**,
/ # *pour une CHARGE_FACE*
/ **'GROUP_FACE'**,
/ **'FACE'**,
/ # *pour une CHARGE_ARETE*

```
                                / 'GROUP_ARETE',
                                / 'ARETE',
                              / # pour une CHARGE_NOEUD
                                / 'NOEUD',
                    ♦ NOMS = noms, [list<str>]
                    ), # fin LIEU
  ), # fin AFFECT
] # fin liste
```

Ainsi, dans le contexte de la classe **CHARGEMENT**, la propriété doit être un objet de type **CHARGE**.

Néanmoins, la classe **CHARGE** est une classe abstraite qui se dérive en plusieurs classes elles-mêmes abstraites :

- **CHARGE_NOEUD** pour des charges appliquées sur des noeuds,
- **CHARGE_ARETE** pour des charges appliquées sur des arêtes,
- **CHARGE_FACE** pour des charges appliquées sur des faces,
- **CHARGE_ELEMENT** pour des charges appliquées sur des éléments.

On instancie donc en fait :

- des classes dérivées de **CHARGE_NOEUD** :
  - **FORC_NOEU**.

  La propriété ne peut alors être appliquée que sur des entités de type **'NOEUD'**.
- des classes dérivées de **CHARGE_ARETE** :
  - **FORC_ARETE_2D**,
  - **PRESS_ARETE_2D**,
  - **CISAIL_ARETE_2D**,
  - **PRESS_HYDRO_2D**,
  - **EXCAV_2D**,
  - **FLUX_ECHANG_LINE_2D**.

  La propriété ne peut alors être appliquée que sur des entités de type **'ARETE'** ou **'GROUP_ARETE'**.
- des classes dérivées de **CHARGE_FACE** :
  - **FORC_FACE_3D**,
  - **PRESS_FACE_3D**,
  - **PRESS_HYDRO_3D**,
  - **EXCAV_3D**,
  - **FLUX_ECHANG_LINE_3D**.

  La propriété ne peut alors être appliquée que sur des entités de type **'FACE'** ou **'GROUP_FACE'**.
- des classes dérivées de **CHARGE_ELEMENT** :
  - **POIDS**,
  - **FORC_MAILLE_COQ**,
  - **FORC_MAILLE_POUT_2D**,
  - **FORC_MAILLE_POUT_3D**,
  - **CONTR_UNIF**,
  - **CONTR_GEOSTAT**.

  La propriété ne peut alors être appliquée que sur des entités de type **'MAILLE'** ou **'GROUP_MAILLE'**.

2.15.3 Classe FORC_NOEU

La classe **FORC_NOEU** permet de définir un chargement de type forces nodales généralisées (forces ou couples).

fnoeud = **FORC_NOEU**(
♦ | **FX** = fx, [**str**]
| **FY** = fy, [**float**]
| **FZ** = fz, [**float**]
| **MX** = mx, [**float**]
| **MY** = my, [**float**]
| **MZ** = mz, [**float**]
| **FREL** = frel, [**float**]
) # *fin FORC_NOEU*

FX : réel donnant la force f_x dans le repère du maillage.
FY : réel donnant la force f_y dans le repère du maillage.
FZ : réel donnant la force f_z dans le repère du maillage.
MX : réel donnant le couple m_x dans le repère du maillage.
MY : réel donnant le couple m_y dans le repère du maillage.
MZ : réel donnant le couple m_z dans le repère du maillage.
FREL : réel donnant l'action duale $frel$ du déplacement relatif $drel$ dans l'axe d'une barre frottante.

2.15.4 Classe FORC_ARETE_2D

La classe **FORC_ARETE_2D** permet de définir un chargement de type force linéique sur des arêtes d'éléments plans.

farete2d = **FORC_ARETE_2D**(
♦ | **FX** = fx, [**float**]
| **FY** = fy, [**float**]
) # *fin FORC_ARETE_2D*

FX : réel donnant la force f_x dans le repère du maillage.
FY : réel donnant la force f_y dans le repère du maillage.

2.15.5 Classe PRESS_ARETE_2D

La classe **PRESS_ARETE_2D** permet de définir un chargement de type pression linéique sur des arêtes d'éléments plans.

parete2d = **PRESS_ARETE_2D**(
♦ **P** = p, [**float**]
) # *fin PRESS_ARETE_2D*

P : réel donnant la pression.

2.15.6 Classe CISAIL_ARETE_2D

La classe **CISAIL_ARETE_2D** permet de définir un chargement de type cisaillement linéique sur des arêtes d'éléments plans.

carete2d = **CISAIL_ARETE_2D**(
♦ **C** = c, [**float**]
) # *fin CISAIL_ARETE_2D*

C : réel donnant le cisaillement.

2.15.7 Classe PRESS_HYDRO_2D

La classe **PRESS_HYDRO_2D** permet de définir un chargement de type pression hydrostatique sur des arêtes d'éléments plans.

phydro2d = **PRESS_HYDRO_2D**(
♦ **COTE_SURF** = cote, [**float**]
♦ **POIDS_VOL** = poids, [**float**]
♦ **VECT_VERTI** = vect, [**list<float>**]
) # *fin PRESS_HYDRO_2D*

COTE_SURF : réel donnant la cote de la surface libre.
POIDS_VOL : réel donnant le poids volumique du fluide.
VECT_VERTI : liste de 2 réels donnant les coordonnées du vecteur unitaire orienté suivant la verticale ascendante.

2.15.8 Classe FLUX_ECHANG_LINE_2D

La classe **FLUX_ECHANG_LINE_2D** permet de définir un chargement de type flux d'échange linéaire sur des arêtes d'éléments plans.

fech2d = **FLUX_ECHANG_LINE_2D**(
♦ **TETAE** = tetae, [**float**]
) # *fin FLUX_ECHANG_LINE_2D*

TETAE : réel donnant la valeur de la température extérieure.

2.15.9 Classe FLUX_ECHANG_LINE_3D

La classe **FLUX_ECHANG_LINE_3D** permet de définir un chargement de type flux d'échange linéaire sur des faces d'éléments volumiques.

fech3d = **FLUX_ECHANG_LINE_3D**(
♦ **TETAE** = tetae, [**float**]
) # *fin FLUX_ECHANG_LINE_3D*

TETAE : réel donnant la valeur de la température extérieure.

2.15.10 Classe EXCAV_2D

La classe **EXCAV_2D** permet de définir un chargement de type forces de déconfinement pour les éléments de massif bidimensionnel.

```
excav2d = EXCAV_2D(
          ♦ MASSIF_REST = massif_rest, [list<LIEU>]
          ♦ / PROFIL = profil, [list<COUCHE_SOL>]
            / NOM_RESOL = nom, [str]
          ) # fin EXCAV_2D
```

MASSIF_REST : liste d'objets du type **LIEU** définissant les zones restantes du maillage après excavation.
PROFIL : liste d'objets de type **COUCHE_SOL** définissant un profil de sol permettant de déterminer un état de contraintes initial avant excavation.
NOM_RESOL : chaîne de caractères donnant le nom de la résolution ayant conduit à l'état de contraintes initial avant excavation.

2.15.11 Classe EXCAV_3D

La classe **EXCAV_3D** permet de définir un chargement de type forces de déconfinement pour les éléments de massif tridimensionnel.

```
excav3d = EXCAV_3D(
          ♦ MASSIF_REST = massif_rest, [list<LIEU>]
          ♦ / PROFIL = profil, [list<COUCHE_SOL>]
            / NOM_RESOL = nom, [str]
          ) # fin EXCAV_3D
```

MASSIF_REST : liste d'objets du type **LIEU** définissant les zones restantes du maillage après excavation.
PROFIL : liste d'objets de type **COUCHE_SOL** définissant un profil de sol permettant de déterminer un état de contraintes initial avant excavation.
NOM_RESOL : chaîne de caractères donnant le nom de la résolution ayant conduit à l'état de contraintes initial avant excavation.

2.15.12 Classe DECONFIN_2D

La classe **DECONFIN_2D** permet de définir un chargement de type forces de déconfinement pour un calcul bidimensionnel.

```
deconf2d = DECONFIN_2D(
           ♦ MASSIF = massif, [list<LIEU>]
           ♦ TYP_MASSIF = / 'REST',
                          / 'EXCAV',
           ♦ LIEU_AUTO = / 'OUI',
                         / 'NON',
           ♦ AVEC_PESANT = / 'OUI',
                             ♦ POIDS_VOL = pvol, [list<float>]
```

```
                        ◊ ACCEL_PESANT = accel, [list<float>]
                    / 'NON',
♦ NOM_BASE = nom, [str]
) # fin DECONFIN_2D
```

MASSIF : liste d'objets du type **LIEU** définissant les zones excavées ou restantes.
TYP_MASSIF : chaîne de caractères indiquant si la zone spécifiée dans **MASSIF** est une zone restante (**'REST'**) ou excavée (**'EXCAV'**).
LIEU_AUTO : chaîne de caractères indiquant si le solveur CESAR détermine automatiquement (**'OUI'**) ou pas (**'NON'**) la zone à charger.
Remarque : dans le cas où **LIEU_AUTO** vaut **'OUI'**, l'utilisateur doit quand même affecter le chargement sur un **LIEU** quelconque, mais qui ne sera pas pris en considération.
AVEC_PESANT : chaîne de caractères indiquant si le chargement doit prendre en compte la pesanteur (**'OUI'**) ou pas (**'NON'**).
POIDS_VOL : liste de réels donnant les valeurs des poids volumiques à prendre en compte.
ACCEL_PESANT : liste de 2 réels donnant les composantes du vecteur des forces massiques.
NOM_BASE : chaîne de caractères donnant le nom de la base de résultats à exploiter pour calculer le chargement.

2.15.13 Classe DECONFIN_3D

La classe **DECONFIN_3D** permet de définir un chargement de type forces de déconfinement pour un calcul tridimensionnel.

```
deconf3d = DECONFIN_3D(
        ♦ MASSIF = massif, [list<LIEU>]
        ♦ TYP_MASSIF = / 'REST',
                       / 'EXCAV',
        ♦ LIEU_AUTO = / 'OUI',
                      / 'NON',
        ♦ AVEC_PESANT = / 'OUI',
                          ♦ POIDS_VOL = pvol, [list<float>]
                          ◊ ACCEL_PESANT = accel, [list<float>]
                        / 'NON',
        ♦ NOM_BASE = nom, [str]
        ) # fin DECONFIN_3D
```

MASSIF : liste d'objets du type **LIEU** définissant les zones excavées ou restantes.
TYP_MASSIF : chaîne de caractères indiquant si la zone spécifiée dans **MASSIF** est une zone restante (**'REST'**) ou excavée (**'EXCAV'**).
LIEU_AUTO : chaîne de caractères indiquant si le solveur CESAR détermine automatiquement (**'OUI'**) ou pas (**'NON'**) la zone à charger.
Remarque : dans le cas où **LIEU_AUTO** vaut **'OUI'**, l'utilisateur doit quand même affecter le chargement sur un **LIEU** quelconque, mais qui ne sera pas pris en considération.
AVEC_PESANT : chaîne de caractères indiquant si le chargement doit prendre en compte la pesanteur (**'OUI'**) ou pas (**'NON'**).
POIDS_VOL : liste de réels donnant les valeurs des poids volumiques à prendre en compte.
ACCEL_PESANT : liste de 3 réels donnant les composantes du vecteur des forces massiques.

NOM_BASE : chaîne de caractères donnant le nom de la base de résultats à exploiter pour calculer le chargement.

2.15.14 Classe FORC_FACE_3D

La classe **FORC_FACE_3D** permet de définir un chargement de type force surfacique sur des faces d'éléments volumiques.

fface3d = **FORC_FACE_3D**(
♦ | **FX** = fx, [**float**]
| **FY** = fy, [**float**]
| **FZ** = fz, [**float**]
) # *fin FORC_FACE_3D*

FX : réel donnant la force f_x dans le repère du maillage.
FY : réel donnant la force f_y dans le repère du maillage.
FZ : réel donnant la force f_z dans le repère du maillage.

2.15.15 Classe PRESS_FACE_3D

La classe **PRESS_FACE_3D** permet de définir un chargement de type pression sur des faces d'éléments volumiques.

pface3d = **PRESS_FACE_3D**(
♦ **P** = p, [**float**]
) # *fin PRESS_FACE_3D*

P : réel donnant la pression.

2.15.16 Classe PRESS_HYDRO_3D

La classe **PRESS_HYDRO_3D** permet de définir un chargement de type pression hydrostatique sur des faces d'éléments volumiques.

phydro3d = **PRESS_HYDRO_3D**(
♦ **COTE_SURF** = cote, [**float**]
♦ **POIDS_VOL** = poids, [**float**]
♦ **VECT_VERTI** = vect, [**list<float>**]
) # *fin PRESS_HYDRO_3D*

COTE_SURF : réel donnant la cote de la surface libre.
POIDS_VOL : réel donnant le poids volumique du fluide.
VECT_VERTI : liste de 3 réels donnant les coordonnées du vecteur unitaire orienté suivant la verticale ascendante.

2.15.17 Classe POIDS

La classe **POIDS** permet de définir un chargement de type poids ou plus généralement force volumique sur des éléments volumiques.

poids = **POIDS**(
♦ **INTENS** = intens [**float**]
♦ **VECT_DIR** = vect_dir, [**list<float>**]
) # *fin POIDS*

INTENS : réel donnant l'intensité de la force volumique.
VECT_DIR : liste de 3 réels donnant la direction de la force volumique.

2.15.18 Classe FORC_MAILLE_COQ

La classe **FORC_MAILLE_COQ** permet de définir un chargement de type force généralisée (force ou couple) surfacique sur des éléments de coque.

fmailcoq = **FORC_MAILLE_COQ**(
♦ | **FX** = fx, [**float**]
| **FY** = fy, [**float**]
| **FZ** = fz, [**float**]
| **MX** = mx, [**float**]
| **MY** = my, [**float**]
| **MZ** = mz, [**float**]
) # *fin FORC_MAILLE_COQ*

FX : réel donnant la force f_x dans le repère du maillage.
FY : réel donnant la force f_y dans le repère du maillage.
FZ : réel donnant la force f_z dans le repère du maillage.
MX : réel donnant le couple m_x dans le repère du maillage.
MY : réel donnant le couple m_y dans le repère du maillage.
MZ : réel donnant le couple m_z dans le repère du maillage.

2.15.19 Classe FORC_MAILLE_POUT_2D

La classe **FORC_MAILLE_POUT_2D** permet de définir un chargement de type force généralisée (force ou couple) linéique sur des éléments de poutre bidimensionnelle.

fmailpou2d = **FORC_MAILLE_POUT_2D**(
♦ | **FX** = fx, [**float**]
| **FY** = fy, [**float**]
| **MZ** = mz, [**float**]
) # *fin FORC_MAILLE_POUT_2D*

FX : réel donnant la force f_x dans le repère du maillage.
FY : réel donnant la force f_y dans le repère du maillage.
MZ : réel donnant le couple m_z dans le repère du maillage.

2.15.20 Classe FORC_MAILLE_POUT_3D

La classe **FORC_MAILLE_POUT_3D** permet de définir un chargement de type force généralisée (force ou couple) linéique sur des éléments de poutre tridimensionnelle.

```
fmailpou3d = FORC_MAILLE_POUT_3D(
               ♦ | FX = fx, [float]
                 | FY = fy, [float]
                 | FZ = fz, [float]
                 | MX = mx, [float]
                 | MY = my, [float]
                 | MZ = mz, [float]
               ) # fin FORC_MAILLE_POUT_3D
```

FX : réel donnant la force f_x dans le repère du maillage.
FY : réel donnant la force f_y dans le repère du maillage.
FZ : réel donnant la force f_z dans le repère du maillage.
MX : réel donnant le couple m_x dans le repère du maillage.
MY : réel donnant le couple m_y dans le repère du maillage.
MZ : réel donnant le couple m_z dans le repère du maillage.

2.15.21 Classe CONTR_UNIF

La classe **CONTR_UNIF** permet de définir un chargement de type contraintes initiales uniformes. Par défaut, ces contraintes initiales sont non seulement introduites en tant qu'état initial de contraintes mais sont également prises en compte dans le second membre du calcul.

```
contr_unif = CONTR_UNIF(
               ♦ TYP_COMP = / '2D',
                              ♦ | SXX = sxx, [float]
                                | SYY = syy, [float]
                                | SZZ = szz, [float]
                                | SXY = sxy, [float]
                            / 'AXISYM',
                              ♦ | SXX = sxx, [float]
                                | SYY = syy, [float]
                                | SZZ = szz, [float]
                                | SXY = sxy, [float]
                            / '3D',
                              ♦ | SXX = sxx, [float]
                                | SYY = syy, [float]
                                | SZZ = szz, [float]
                                | SXY = sxy, [float]
                                | SYZ = syz, [float]
                                | SZX = sxz, [float]
               ◊ MODE = / 'INIT_SEUL',
                        / 'CHARG_SEUL',
               ) # fin CONTR_UNIF
```

TYP_COMP : chaîne de caractère indiquant le type de modèle utilisé à choisir parmi :
— **'2D'** : modèle bidimensionnel,
— **'AXISYM'** : modèle axisymétrique,
— **'3D'** : modèle tridimensionnel.
SXX : réel donnant la valeur de la composante σ_{xx} du tenseur des contraintes.
SYY : réel donnant la valeur de la composante σ_{yy} du tenseur des contraintes.
SZZ : réel donnant la valeur de la composante σ_{zz} du tenseur des contraintes.
SXY : réel donnant la valeur de la composante σ_{xy} du tenseur des contraintes.
SYZ : réel donnant la valeur de la composante σ_{yz} du tenseur des contraintes.
SXZ : réel donnant la valeur de la composante σ_{xz} du tenseur des contraintes.
MODE : chaîne de caractères permettant de limiter le mode de prise en compte des contraintes initiales :
— **'INIT_SEUL'** : pour une seule initialisation de l'état de contraintes,
— **'CHARG_SEUL'** : pour une seule prise en compte des contraintes initiales dans le chargement.

2.15.22 Classe CONTR_GEOSTAT

La classe **CONTR_GEOSTAT** permet de définir un chargement de type contraintes initiales d'origine géostatique. Par défaut, ces contraintes initiales sont non seulement introduites en tant qu'état initial de contraintes mais sont également prises en compte dans le second membre du calcul.

```
contr_geostat = CONTR_GEOSTAT(
                ♦ PROFIL = profil, [list<COUCHE_SOL>]
                ♦ MODE = / 'INIT_SEUL',
                         / 'CHARG_SEUL',
                ) # fin CONTR_GEOSTAT
```

PROFIL : liste d'objets de type **COUCHE_SOL** définissant le profil du sol.
MODE : chaîne de caractères permettant de limiter le mode de prise en compte des contraintes initiales d'origine géostatique :
— **'INIT_SEUL'** : pour une seule initialisation de l'état de contraintes,
— **'CHARG_SEUL'** : pour une seule prise en compte des contraintes initiales dans le chargement.

Description des couches de sol. La classe **COUCHE_SOL** permet de définir les caractéristiques d'une couche de sol.

```
couche = COUCHE_SOL(
         ♦ COTE_SUP = cote_sup, [float]
         ♦ POIDS_VOL = poids_vol, [float]
         ♦ COEF_POUS_1 = coef_pous_1, [float]
         ♦ COEF_POUS_2 = coef_pous_2, [float]
         ) # fin COUCHE_SOL
```

COTE_SUP : réel donnant la valeur de la cote dans le système d'axes utilisateur de la limite supérieure de la couche de sol.

POIDS_VOL : réel donnant la valeur du poids volumique du sol.
COEF_POUS_1 : réel donnant la valeur du coefficient de poussée latérale suivant Ox.
COEF_POUS_2 : réel donnant la valeur du coefficient de poussée latérale suivant selon Oz pour les calculs en déformation plane ou selon Oy pour les calculs tridimensionnels.

2.16 Données relatives aux analyses

La classe **ANALYSE** est la classe permettant de définir le type d'analyse à réaliser.
Par exemple : analyse statique linéaire ou non linéaire, analyse modale, analyse dynamique, ...
C'est une classe abstraite. Elle constitue la classe de base des classes instanciables suivantes :

- **STAT_LINE**,
- **STAT_NON_LINE**,
- **MODAL_LINE**,
- **DYNA_LINE_TEMP**,
- **DYNA_LINE_HARMO**,
- **THERM_BETON_JEUNE_AGE**,
- **MECA_BETON_JEUNE_AGE**.

2.16.1 Classe STAT_LINE

La classe **STAT_LINE** est la classe permettant de définir les caractéristiques d'une analyse statique linéaire. La caractéristique principale réside dans le choix du solveur linéaire à utiliser.

```
stat_lin = STAT_LINE(
        ♦ NOM = nom, [str]
        ♦ SOLVEUR = / 'MULTI_FRONT',
                    / 'LIGNE_CIEL',
        ) # fin STAT_LINE
```

NOM : chaîne de caractères définissant le nom de l'analyse. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
SOLVEUR : chaîne de caractères définissant le type du solveur linéaire à choisir parmi :

- **'MULTI_FRONT'** : pour le solveur linéaire direct de type multifrontal,
- **'LIGNE_CIEL'** : pour le solveur linéaire direct de type ligne de ciel.

2.16.2 Classe STAT_NON_LINE

La classe **STAT_NON_LINE** est la classe permettant de définir les caractéristiques d'une analyse statique non linéaire. La non linéarité considérée ici est celle relative aux lois de comportement des matériaux. Elle permet de choisir le type de méthode itérative de résolution du problème non linéaire ainsi que le type du solveur linéaire. Le pilotage de la méthode de résolution du problème non linéaire est réalisé en définissant :

- son type (Newton complet, ou modifié, ...) ainsi que le nombre maximal d'itérations à chaque pas via la classe **METHOD_ITER**,
- une incrémentation via la classe **INCREMENTATION**,
- des fonctions multiplicatrices s'appliquant à des chargements ou conditions limites. A noter qu'un chargement ou une condition limite référencé dans une **RESOLUTION** mais

n'étant pas invoqué explicitement par une fonction multiplicatrice se verra affecté implicitement la fonction constante égale à 1.

Dans le cas où l'analyse comporte du contact, elle permet également de définir les critères de contact à vérifier via la classe **VERIF_CONTACT**.

```
stat_nlin = STAT_NON_LINE(
        ♦ NOM = nom, [str]
        ♦ SOLVEUR = / 'MULTI_FRONT',
                    / 'LIGNE_CIEL',
        ♦ METH = meth, [METHOD_ITER]
        ♦ INCREM = increm, [INCREMENTATION]
        ◊ FONC = fonc, [list<FONCT_MULT>]
        ◊ CRITERE = criter, [VERIF_CONTACT]
        ◊ DEPLA_INI_ZERO = / 'OUI',
                           / 'NON',
        ) # fin STAT_NON_LINE
```

NOM : chaîne de caractères définissant le nom de l'analyse. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
SOLVEUR : chaîne de caractères définissant le type du solveur linéaire à choisir parmi :
— **'MULTI_FRONT'** : pour le solveur linéaire direct de type multifrontal,
— **'LIGNE_CIEL'** : pour le solveur linéaire direct de type ligne de ciel.
METH : objet de type **METHOD_ITER** définissant les propriétés de la méthode itérative utilisée.
INCREM : objet de type **INCREMENTATION** définissant les pas de "pseudo temps" à considérer.
FONC : liste d'objets de type **FONCT_MULT** définissant des fonctions multiplicatrices (fonctions du "pseudo temps") s'appliquant aux chargements ou conditions limites.
CRITERE : objet de type **VERIF_CONTACT** permettant de définir les critères de contact à vérifier.
DEPLA_INI_ZERO : chaîne de caractères définissant si les déplacements doivent être remis à zéro (**'OUI'**) ou pas (**'NON'**) lors de la reprise d'un calcul précédent. Ceci permet de récupérer un état de contraintes issu d'un calcul sans récupérer les déplacements associés, d'obtenir lors d'un phasage l'incrément de déplacements dû à chaque phase. Ce mot-clé n'a donc un sens que si la **RESOLUTION** associée a son mot-clé **INIT** renseigné.

Méthode itérative. La classe **METHOD_ITER** est la classe permettant de définir les propriétés de la méthode itérative à utiliser pour la résolution d'un problème non linéaire.

```
meth = METHOD_ITER(
        ♦ TYP = / 'CONTR_INIT',
                / 'NEWTON_MODIF',
                / 'NEWTON_COMPLET',
        ♦ NB_ITER = nb_iter, [int]
        ♦ TOLER = toler, [float]
        ) # fin METHOD_ITER
```

TYP : chaîne de caractères définissant le type de la méthode à choisir parmi :
— **'CONTR_INIT'** : pour la méthode des contraintes initiales,
— **'NEWTON_MODIF'** : pour la méthode de Newton modifié,
— **'NEWTON_COMPLET'** : pour la méthode de Newton complète (calcul systématique de la rigidité tangente).
NB_ITER : entier définissant le nombre maximum d'itérations de la méthode.
TOLER : réel définissant la tolérance relative sur la convergence de la méthode.

Incrémentation. La classe **INCREMENTATION** est la classe permettant de définir l'incrémentation à utiliser pour la résolution d'un problème non linéaire, c'est à dire les valeurs du "pseudo temps" en lesquelles on applique un incrément de chargement ou de condition limite.

```
increm = INCREMENTATION(
        ♦ NOM = nom, [str]
        ♦ LIST_INST = list_inst, [list<float>]
        ) # fin INCREMENTATION
```

NOM : chaîne de caractères définissant le nom de l'incrémentation.
LIST_INST : liste d'entiers définissant les valeurs du "pseudo temps".

Fonction multiplicatrice. La classe **FONCT_MULT** est la classe permettant de définir une fonction du "pseudo temps" à appliquer en tant que fonction multiplicatrice à un chargement ou une condition limite pour la résolution d'un problème non linéaire. Cette fonction est définie de façon discrète en donnant une liste de valeurs de pseudo-temps ainsi que la liste de valeurs des coefficients multiplicateurs correspondants. A noter que les valeurs de pseudo temps ne doivent pas nécessairement correspondre point par point à la liste de valeurs fournis via l'attribut **INCREM** de **STAT_NON_LINE**.

```
fonc = FONCT_MULT(
        ♦ NOM = nom, [str]
        ♦ LIST_INST = list_inst, [list<float>]
        ♦ LIST_COEF = list_coef, [list<float>]
        ♦ / NOM_CAS = nom_cas, [str]
          / NOM_COND = nom_cond, [str]
        ) # fin FONCT_MULT
```

NOM : chaîne de caractères définissant le nom de la fonction multiplicatrice. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
LIST_INST : liste de réels définissant les valeurs du "pseudo temps".
LIST_COEF : liste de réels définissant les valeurs de la fonction (les coefficients).
NOM_CAS : chaîne de caratères donnant le nom du cas de chargement auquel s'applique la fonction.
NOM_COND : chaîne de caratères donnant le nom de la condition limite à laquelle s'applique la fonction.

Critères de contact. La classe **VERIF_CONTACT** est la classe permettant de définir les critères de contact à vérifier.

verif = **VERIF_CONTACT**(
♦ **INTER_PENETR** = / **'OUI'**,
/ **'NON'**,
♦ **DECOL** = / **'OUI'**,
/ **'NON'**,
♦ **FROT** = / **'OUI'**,
/ **'NON'**,
) # *fin VERIF_CONTACT*

INTER_PENETR : chaîne de caractères définissant si le critère de non-interpénétration doit être vérifié (**'OUI'**) ou pas (**'NON'**).
DECOL : chaîne de caractères définissant si le critère de décollement doit être vérifié (**'OUI'**) ou pas (**'NON'**).
FROT : chaîne de caractères définissant si le critère de frottement doit être vérifié (**'OUI'**) ou pas (**'NON'**).

2.16.3 Classe MODAL_LINE

La classe **MODAL_LINE** est la classe permettant de définir les caractéristiques d'une analyse modale linéaire (problème de valeurs propres généralisé $Ax = \lambda Bx$). Elle s'utilise donc notamment pour un problème de vibration propre ou de flambement linéaire. Dans le cas du flambement, elle permet de définir la partie du chargement qui est fixe et celle qui est variable c'est à dire celle à laquelle on applique le coefficient critique de flambement. Le pilotage de la méthode de résolution du problème aux valeurs propres est réalisé à l'aide de la classe **METHOD_VAL_PROP**.

modal = **MODAL_LINE**(
♦ **NOM** = nom, [**str**]
♦ **TYP** = / **'FLAMB'**,
◊ **CAS_CHARG_FIXE** = cas_charg_fix, [**CAS_CHARGEMENT**]
♦ **CAS_CHARG_VARI** = cas_charg_vari, [**CAS_CHARGEMENT**]
/ **'VIBRA'**,
♦ **DECAL_SPECTR** = decal_spectr [**float**]
♦ **NB_VAL_PROP** = nb_val_prop [**int**]
♦ **METH** = meth, [**METHOD_VAL_PROP**]
♦ **SOLVEUR** = / **'MULTI_FRONT'**,
/ **'LIGNE_CIEL'**,
) # *fin MODAL_LINE*

NOM : chaîne de caractères définissant le nom de l'analyse. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
TYP : chaîne de caractères définissant le type du problème modal linéaire à choisir parmi :
— **'FLAMB'** : pour un problème de flambement linéaire,
— **'VIBRA'** : pour un problème de vibration propre.
CAS_CHARG_FIXE : objet de type **CAS_CHARGEMENT** définissant la partie fixe du chargement.

CAS_CHARG_VARI : objet de type **CAS_CHARGEMENT** définissant la partie variable du chargement.
DECAL_SPECTR : réel au voisinage duquel on cherche les valeurs propres.
NB_VAL_PROP : entier donnant le nombre de valeurs propres à calculer.
METH : objet de type **METHOD_VAL_PROP** permettant de piloter la méthode de résolution du problème aux valeurs propres
SOLVEUR : chaîne de caractères définissant le type du solveur linéaire à choisir parmi :
— **'MULTI_FRONT'** : pour le solveur linéaire direct de type multifrontal,
— **'LIGNE_CIEL'** : pour le solveur linéaire direct de type ligne de ciel.

Méthode de résolution du problème aux valeurs propres. La classe **METHOD_VAL_PROP** est la classe permettant de définir les caractéristiques de la méthode de résolution du problème aux valeurs propres.

```
meth = METHOD_VAL_PROP(
    ♦ TYP = / 'SOUS_ESPAC',
              ♦ DIM_SOUS_ESPAC = dim_sous_espac [int]
              ♦ NB_MAX_ITER = nb_max_iter [int]
    ♦ TOLER = toler [float]
    ) # fin METHOD_VAL_PROP
```

TYP : chaîne de caractères définissant le type de la méthode de résolution du problème aux valeurs propres à choisir parmi :
— **'SOUS_ESPAC'** : pour la méthode du sous-espace.
DIM_SOUS_ESPAC : entier donnant la dimension du sous-espace utilisé.
NB_MAX_ITER : entier donnant le nombre maximum d'itérations de sous-espaces.
TOLER : réel donnant la précision relative pour les valeurs propres.

2.16.4 Classe DYNA_LINE_TEMP

La classe **DYNA_LINE_TEMP** est la classe permettant de définir les caractéristiques d'une analyse dynamique linéaire temporelle. Le pilotage de la méthode de résolution du problème dynamique est réalisé en définissant :
— son type (intégration directe ou superposition modale),
— une incrémentation via la classe **INCREMENTATION**,
— des fonctions multiplicatrices s'appliquant à des chargements ou conditions limites. A noter qu'un chargement ou une condition limite référencé dans une **RESOLUTION** mais n'étant pas invoqué explicitement par une fonction multiplicatrice se verra affecté implicitement la fonction constante égale à 1.

Il est possible de définir un amortissement via la classe **AMORTISSEMENT**.
Dans le cas d'une résolution par superposition modale, on doit définir l'analyse modale générant les modes.

```
dyna_temp = DYNA_LINE_TEMP(
              ♦ NOM = nom, [str]
              ♦ METHOD = / 'DIRECTE',
                         / 'MODALE',
                           ♦ ANALYS_MODAL = analys_modal, [MODAL_LINE]
              ♦ INCREM = increm, [INCREMENTATION]
```

◊ **FONC** = fonc, [**list<FONCT_MULT>**]
◊ **AMOR** = amor, [**AMORTISSEMENT**]
) # *fin DYNA_LINE_TEMP*

NOM : chaîne de caractères définissant le nom de l'analyse. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
METHOD : chaîne de caractères définissant la méthode de résolution du problème dynamique :
— **'DIRECTE'** : pour une résolution par intégration directe,
— **'MODALE'** : pour une résolution par superposition modale.
ANALYS_MODAL : objet de type **MODAL_LINE** définissant l'analyse modale permettant de construire la base modale.
INCREM : objet de type **INCREMENTATION** définissant les pas de temps à considérer.
FONC : liste d'objets de type **FONCT_MULT** définissant des fonctions multiplicatrices (fonctions du temps) s'appliquant aux chargements ou conditions limites.
AMOR : objet de type **AMORTISSEMENT** définissant les caractéristiques d'amortissement.

Amortissement. La classe **AMORTISSEMENT** est la classe permettant de définir des caractéristiques d'amortissement.

amor = **AMORTISSEMENT**(
♦ **TYP** = / **'RAYLEIGH'**,
♦ **COEF_RIGI** = coef_rigi, [**float**]
♦ **COEF_MASSE** = coef_masse, [**float**]
/ **'MODAL'**,
♦ **LIST_COEF_CRIT** = list_coef_crit, [**list<float>**]
) # *fin AMORTISSEMENT*

TYP : chaîne de caractères définissant le type d'amortissement :
— **'RAYLEIGH'** : pour un amortissement de type Rayleigh,
— **'MODAL'** : pour un amortissement de type modal.
COEF_RIGI : réel définissant le coefficient de Rayleigh relatif à la matrice de rigidité.
COEF_MASSE : réel définissant le coefficient de Rayleigh relatif à la matrice de masse.
LIST_COEF_CRIT : liste de réels définissant les pourcentages d'amortissement critique associés aux modes propres retenus.

2.16.5 Classe DYNA_LINE_HARMO

La classe **DYNA_LINE_HARMO** est la classe permettant de définir les caractéristiques d'une analyse dynamique linéaire harmonique en variable complexe.
Il est possible de définir un amortissement via la classe **AMORTISSEMENT**.

dyna_harmo = **DYNA_LINE_HARMO**(
♦ **NOM** = nom, [**str**]
♦ **FREQ** = freq [**float**]
◊ **AMOR** = increm, [**AMORTISSEMENT**]
♦ **CAS_CHARG_REEL** = cas_charg_reel, [**CAS_CHARGEMENT**]

◊ **CAS_CHARG_IMAGIN** = cas_charg_imagin, [**CAS_CHARGEMENT**]
) # *fin DYNA_LINE_HARMO*

NOM : chaîne de caractères définissant le nom de l'analyse. Pour Aster, cette chaîne doit être limitée à 8 caractères car elle est utilisée pour définir un nom de "concept" (au sens d'Aster) dans le fichier de commandes Aster généré automatiquement par le Pilote.
FREQ : réel donnant la fréquence de la sollicitation.
AMOR : objet de type **AMORTISSEMENT** définissant les caractéristiques d'amortissement.
CAS_CHARG_REEL : objet de type **CAS_CHARGEMENT** définissant la partie réelle du chargement.
CAS_CHARG_IMAGIN : objet de type **CAS_CHARGEMENT** définissant la partie imaginaire du chargement.

2.16.6 Classe THERM_BETON_JEUNE_AGE

La classe **THERM_BETON_JEUNE_AGE** est la classe permettant de définir les caractéristiques d'une analyse thermique du béton au jeune âge.

therm_bja = **THERM_BETON_JEUNE_AGE**(
♦ **NOM** = nom, [**str**]
♦ **SOLVEUR** = / **'MULTI_FRONT'**,
/ **'LIGNE_CIEL'**,
♦ **METH** = meth, [**METHOD_ITER**]
♦ **INCREM** = increm, [**INCREMENTATION**]
♦ **ESSAI** = essai, [**ESSAI_QAB**]
◊ **FONC** = fonc, [**list<FONCT_MULT>**]
◊ **TEMP_INIT** = temp, [**float**]
) # *fin THERM_BETON_JEUNE_AGE*

NOM : chaîne de caractères définissant le nom de l'analyse.
SOLVEUR : chaîne de caractères définissant le type du solveur linéaire à choisir parmi :
— **'MULTI_FRONT'** : pour le solveur linéaire direct de type multifrontal,
— **'LIGNE_CIEL'** : pour le solveur linéaire direct de type ligne de ciel.
METH : objet de type **METHOD_ITER** définissant les propriétés de la méthode itérative utilisée.
INCREM : objet de type **INCREMENTATION** définissant les pas de temps à considérer.
ESSAI : objet de type **ESSAI_QAB** définissant les résultats de l'essai QAB.
FONC : liste d'objets de type **FONCT_MULT** définissant des fonctions multiplicatrices (fonctions du temps) s'appliquant aux chargements ou conditions limites.
TEMP_INIT : réel donnant la valeur initiale uniforme de la température.

Essai QAB. La classe **ESSAI_QAB** est la classe permettant de définir les résultats d'un essai QAB.

essai = **ESSAI_QAB**(
♦ **MESURES** = mesures, [**list<MESURE_QAB>**]
♦ **COEF_DEPERD_ABC** = coefs, [**list<float>**]
♦ **CAPAC_CALOR** = capacite, [**float**]

♦ **CSTE_ARRHENIUS** = cste, [**float**]
) # *fin ESSAI_QAB*

MESURES : liste d'objets de type **MESURE_QAB** définissant les mesures de l'essai QAB.
COEF_DEPERD_ABC : liste de 3 réels donnant les coefficients caractéristiques A, B, C des déperditions thermiques du calorimètre.
CAPAC_CALOR : réel donnant la capacité calorifique de l'échantillon.
CSTE_ARRHENIUS : réel donnant la constante de la loi d'Arrhénius.

mesure = **MESURE_QAB**(
♦ **INST** = inst, [**float**]
♦ **THETA** = theta, [**float**]
♦ **THETA_EXT** = thetae, [**float**]
) # *fin MESURE_QAB*

INST : réel donnant la valeur du temps au moment de la mesure.
THETA : réel donnant la valeur de l'échantillon au moment de la mesure.
THETA_EXT : réel donnant la valeur de la température à l'extérieur du calorimètre au moment de la mesure.

2.16.7 Classe MECA_BETON_JEUNE_AGE

La classe **MECA_BETON_JEUNE_AGE** est la classe permettant de définir les caractéristiques d'une analyse mécanique du béton au jeune âge. Elle fait suite à une analyse thermique préalable de type **THERM_BETON_JEUNE_AGE**.

meca_bja = **MECA_BETON_JEUNE_AGE**(
♦ **NOM** = nom, [**str**]
♦ **SOLVEUR** = / **'MULTI_FRONT'**,
/ **'LIGNE_CIEL'**,
♦ **NUM_INST_THERM** = num, [**list<int>**]
♦ **NOM_RESOL_THERM** = nom, [**str**]
) # *fin MECA_BETON_JEUNE_AGE*

NOM : chaîne de caractères définissant le nom de l'analyse.
SOLVEUR : chaîne de caractères définissant le type du solveur linéaire à choisir parmi :
— **'MULTI_FRONT'** : pour le solveur linéaire direct de type multifrontal,
— **'LIGNE_CIEL'** : pour le solveur linéaire direct de type ligne de ciel.
NUM_INST_THERM : liste d'entiers définissant les numéros des instants de l'analyse thermique réalisée préalablement pour lesquels on veut réaliser le calcul mécanique.
NOM_RESOL_THERM : chaîne de caractères définissant le nom de la résolution associée à l'analyse thermique réalisée préalablement.

2.17 Données relatives aux post-traitements

2.17.1 Classe POST_TRAITEMENT

La classe **POST_TRAITEMENT** est la classe permettant de définir certaines tâches de post-traitement qui seront à réaliser pour une **RESOLUTION**.

post = **POST_TRAITEMENT**(
♦ **NOM** = nom, [**str**]
◊ **ESTIM** = estim, [**ESTIM_ERREUR**]
) # *fin POST_TRAITEMENT*

NOM : chaîne de caractères définissant le nom du post-traitement.
ESTIM : objet de type **ESTIM_ERREUR** définissant les propriétés d'un calcul d'estimation d'erreur a posteriori.

Estimation d'erreur a posteriori. La classe **ESTIM_ERREUR** est la classe permettant de définir un calcul d'estimation d'erreur a posteriori. Actuellement, seul l'estimateur d'erreur au sens de Zhu-Zienkiewicz est disponible et son utilisation est limitée à des modèles élastiques linéaires en élasticité plane.

post = **ESTIM_ERREUR**(
♦ **TYP** = / **'ZZ2'**,
) # *fin ESTIM_ERREUR*

TYP : chaîne de caractères définissant le type d'estimateur d'erreur :
— **'ZZ2'** : pour l'estimateur d'erreur de Zhu-Zienkiewicz (version de 1992).

2.18 Données relatives aux résolutions

2.18.1 Classe RESOLUTION

La classe **RESOLUTION** est la classe permettant de définir un cas d'étude c'est à dire l'ensemble des données nécessaires pour pouvoir appeler un solveur éléments finis (CESAR, ASTER, ...). Elle contient donc les informations sur le maillage, les modèles, les matériaux, les conditions limites, les cas de chargements, le type d'analyse. En pratique, ses attributs sont donc renseignés par des références à des objets renseignés préalablement de type :

— **MODELE**,
— **MATERIAU**,
— **CARACTERISTIQUE**,
— **LIAISON**,
— **COND_LIMITE**,
— **CAS_CHARGEMENT**,
— **ANALYSE**,
— **POST_TRAITEMENT**.

Elle permet également d'initialiser le calcul avec des valeurs issues d'un calcul précédent ou bien de stocker les résultats du calcul via la classe **BASE**.

resol = **RESOLUTION**(
♦ **NOM** = nom, [**str**]
♦ **MODEL** = mod, [**MODELE**]
♦ **MATER** = mat, [**MATERIAU**]
◊ **CARAC** = cara, [**CARACTERISTIQUE**]
◊ **LIAIS** = liais, [**LIAISON**]
◊ **COND_LIM** = cond, [**COND_LIMITE**]

◊ **CAS_CHARG** = cas, [**list<CAS_CHARGEMENT>**]
♦ **ANALYS** = ana, [**ANALYSE**]
◊ **INIT** = init, [**BASE**]
◊ **STOCK** = stock, [**BASE**]
◊ **POST** = post, [**POST_TRAITEMENT**]
) # *fin RESOLUTION*

NOM : chaîne de caractères définissant le nom de la résolution.
MODEL : objet de type **MODELE** définissant les propriétés relatives aux modèles.
MATER : objet de type **MATERIAU** définissant les propriétés relatives aux matériaux.
CARAC : objet de type **CARACTERISTIQUE** définissant les propriétés relatives aux caractéristiques.
LIAIS : objet de type **LIAISON** définissant les propriétés relatives aux liaisons.
CAS_CHARG : objet de type **CAS_CHARGEMENT** définissant les propriétés relatives aux cas de chargements.
COND_LIM : objet de type **COND_LIMITE** définissant les propriétés relatives aux conditions limites.
ANALYS : objet de type **ANALYSE** définissant les propriétés relatives au type d'analyse.
INIT : objet de type **BASE** définissant une base de résultats permettant d'initialiser le calcul.
STOCK : objet de type **BASE** définissant une base de stockage des résultats du calcul.
POST : objet de type **POST_TRAITEMENT** définissant un calcul de post-traitement.

Base de résultats. La classe **BASE** est la classe permettant de définir une base de résultats. Cette base est définie par un nom et se matérialise concrètement par un fichier du même nom. Elle correspond par exemple aux fichiers, souvent suffixés .rst, générés ou lus par CESAR. Une résolution peut exporter ses résultats dans une base ou bien s'initialiser par lecture d'une base générée par une autre résolution.

base = **BASE**(
♦ **NOM** = nom, [**str**]
) # *fin BASE*

NOM : chaîne de caractères définissant le nom de la base.

2.18.2 Classe PHASE

La classe **PHASE** est la classe permettant de définir une phase d'un phasage de construction. Elle dérive de la classe **RESOLUTION**. Son constructeur est très comparable à celui de sa classe mère.
La spécificité d'une **PHASE** réside dans sa donnée **ACTIV** qui permet de définir une **ACTIVATION**, c'est à dire l'ensemble des groupes d'éléments au sens de CESAR qui doivent être "activés" au cours la phase.

phase = **PHASE**(
♦ **NOM** = nom, [**str**]
♦ **ACTIV** = activ, [**ACTIVATION**]
♦ **MATER** = mat, [**MATERIAU**]

◊ **CARAC** = cara, [**CARACTERISTIQUE**]
◊ **LIAIS** = liais, [**LIAISON**]
◊ **COND_LIM** = cond, [**COND_LIMITE**]
◊ **CAS_CHARG** = cas, [**list<CAS_CHARGEMENT>**]
♦ **ANALYS** = ana, [**ANALYSE**]
◊ **INIT** = init, [**BASE**]
◊ **STOCK** = stock, [**BASE**]
◊ **POST** = post, [**POST_TRAITEMENT**]
) # *fin PHASE*

NOM : chaîne de caractères définissant le nom de la résolution.
ACTIV : objet de type **ACTIVATION** définissant les propriétés relatives aux activations d'éléments.
MATER : objet de type **MATERIAU** définissant les propriétés relatives aux matériaux.
CARAC : objet de type **CARACTERISTIQUE** définissant les propriétés relatives aux caractéristiques.
CAS_CHARG : objet de type **CAS_CHARGEMENT** définissant les propriétés relatives aux cas de chargements.
LIAIS : objet de type **LIAISON** définissant les propriétés relatives aux liaisons.
COND_LIM : objet de type **COND_LIMITE** définissant les propriétés relatives aux conditions limites.
ANALYS : objet de type **ANALYSE** définissant les propriétés relatives au type d'analyse.
INIT : objet de type **BASE** définissant une base de résultats permettant d'initialiser le calcul.
STOCK : objet de type **BASE** définissant une base de stockage des résultats du calcul.
POST : objet de type **POST_TRAITEMENT** définissant un calcul de post-traitement.

2.19 Données relatives aux résultats

2.19.1 Classe RESULTAT

La classe **RESULTAT** est la classe permettant d'importer ou d'exporter un ensemble de résultats de calcul. Le fichier de résultats (en entrée ou en sortie) est défini via la classe **FICHIER**. Il est possible de filtrer les résultats issus du calcul via la classe **FILTRE**. Parmi les résultats filtrés, on peut de plus définir des sélections de résultats via la classe **SELECT**.

resu = **RESULTAT**(
♦ **NOM** = nom, [**str**]
◊ **RESOL** = reso, [**RESOLUTION**]
◊ **FILTRE** = filtre, [**list<FILTRE>**]
◊ **FIC_IN** = **FICHIER**(
♦ **FORMAT** = / **'CESAR'**,
♦ **NOM** = nom, [**str**]
), # *fin FICHIER*
◊ **FIC_OUT** = **FICHIER**(
♦ **FORMAT** = / **'GMSH'**,
/ **'MED'**,
♦ **NOM** = nom, [**str**]

```
                ), # fin FICHIER
◊ SELECT = select, [list<SELECT>]
) # fin RESULTAT
```

NOM : chaîne de caractères définissant le nom du maillage.
RESOL : objet de type **RESOLUTION** donnant la résolution dont relèvent les résultats.
FILTRE : liste d'objets de type **FILTRE** donnant la liste des filtres de résultats.
FIC_IN : objet de type **FICHIER** définissant le fichier de résultat à importer. Actuellement, seul le format **'CESAR'** est disponible.
FIC_OUT : objet de type **FICHIER** définissant le fichier de maillage à exporter. Actuellement, les formats **'GMSH'** et **'MED'** sont disponibles.
SELECT : liste d'objets de type **SELECT** donnant la liste des sélections de résultats. On précise ci-dessous l'utilisation de la classe **SELECT**.

```
filtre = [ # debut liste
    FILTRE(
    ♦ NOM = nom, [str]
    ♦ NOM_CHAMP = / 'DEPLA_2D',
                  / 'DEPLA_3D',
                  / 'DEPLA_2D_ROTA',
                  / 'DEPLA_3D_ROTA',
                  / 'ACCEL_2D',
                  / 'ACCEL_3D',
                  / 'CONTR_2D',
                  / 'CONTR_3D',
                  / 'DEFORM_TOTAL_2D',
                  / 'DEFORM_TOTAL_3D',
                  / 'EFFORT_BAR_2D',
                  / 'EFFORT_BAR_3D',
                  / 'EFFORT_POUTR_2D',
                  / 'EFFORT_POUTR_3D',
                  / 'TEMP',
                  / 'GRAD_TEMP_2D',
                  / 'GRAD_TEMP_3D',
                  / 'VITES_TEMP_2D',
                  / 'VITES_TEMP_3D',
    ◊ / NUM_INST = num_inst, [list<int>]
      / NUM_INCREM = num_increm, [list<int>]
      / NUM_MODE = num_mode, [list<int>]
      / NOM_CAS = nom_cas, [list<str>]
    ), # fin FILTRE
  ] # fin liste
```

NOM : chaîne de caractères définissant le nom du filtre.
NOM_CHAMP : type du champ à choisir parmi **'DEPLA_2D'**, **'DEPLA_3D'**, **'DEPLA_2D_ROTA'**, **'DEPLA_3D_ROTA'**, **'ACCEL_2D'**, **'ACCEL_3D'**, **'CONTR_2D'**, **'CONTR_3D'**, **'DEFORM_TOTAL_2D'**, **'DEFORM_TOTAL_3D'**, **'EFFORT_BAR_2D'**, **'EFFORT_BAR_3D'**, **'EFFORT_POUTR_2D'**, **'EFFORT_POUTR_3D'**, **'TEMP'**, **'GRAD_TEMP_2D'**, **'GRAD_TEMP_3D'**, **'VITES_TEMP_2D'**, **'VITES_TEMP_3D'**.

NUM_INST : liste d'entiers spécifiant les numéros des instants du calcul.
NUM_INCREM : liste d'entiers spécifiant les numéros des incréments du calcul.
NUM_MODE : liste d'entiers spécifiant les numéros des modes du calcul.
NOM_CAS : liste de chaînes de caractères spécifiant les noms des cas de charges du calcul.

```
select = [ # debut liste
          SELECT(
          ♦ FICHIER = FICHIER(
                        ♦ FORMAT = / 'TXT',
                        ♦ NOM = nom, [str]
                        ), # fin FICHIER
          ♦ EXTRACT = extract, [list<EXTRACT>]
          ), # fin SELECT
        ] # fin liste
```

FICHIER : objet de type **FICHIER** définissant le fichier d'export de la sélection de résultats. Actuellement, seul le format **'TXT'** est disponible.
EXTRACT : liste d'objets de type **EXTRACT** définissant les résultats à extraire. On précise ci-dessous l'utilisation autorisée de la classe **EXTRACT**.

```
extract = [ # debut liste
           EXTRACT(
           ♦ GRANDEUR = grandeur, [GRANDEUR]
           ♦ LIEU = LIEU(
                      ♦ TYP = / 'GROUP_MAILLE',
                              / 'GROUP_FACE',
                              / 'GROUP_ARETE',
                              / 'MAILLE',
                              / 'FACE',
                              / 'ARETE',
                      ♦ NOMS = noms, [list<str>]
                      ), # fin LIEU
           ), # fin EXTRACT
         ] # fin liste
```

Ainsi, dans le contexte de la classe **SELECT** :
- la grandeur doit être un objet de type **GRANDEUR**,
- la grandeur ne peut être appliquée que sur des entités de type :
  - **'MAILLE'**,
  - **'FACE'**,
  - **'ARETE'**,
  - **'GROUP_MAILLE'**,
  - **'GROUP_FACE'**,
  - **'GROUP_ARETE'**.

Néanmoins, la classe **GRANDEUR** est une classe abstraite et on ne l'instancie pas. On invoque en fait les classes instanciables suivantes qui en dérivent :
- **DEPLA_2D**,
- **DEPLA_3D**,

- **DEPLA_2D_ROTA**,
- **DEPLA_3D_ROTA**,
- **ACCEL_2D**,
- **ACCEL_3D**,
- **CONTR_2D**,
- **CONTR_3D**,
- **REAC_2D**,
- **REAC_3D**,
- **'DEFORM_TOTAL_2D'**,
- **'DEFORM_TOTAL_3D'**,
- **'EFFORT_BAR_2D'**,
- **'EFFORT_BAR_3D'**,
- **'EFFORT_POUTR_2D'**,
- **'EFFORT_POUTR_3D'**,
- **'TEMP'**,
- **'GRAD_TEMP_2D'**,
- **'GRAD_TEMP_3D'**,
- **'VITES_TEMP_2D'**,
- **'VITES_TEMP_3D'**.

Autres attributs

fields : liste d'objets de type **FIELD_** stockant les champs de résultats au format MED Mémoire.

Méthodes

extraire() : méthode permettant d'extraire du résultat les composantes d'un champ de grandeur en un **LIEU**. Elle possède 3 arguments à passer via les 3 mots-clés suivants :

. **GRANDEUR** : le champ de grandeur à extraire.
. **LIEU** : le lieu de l'extraction.
. **PAS** : entier spécifiant le pas ou incrément de calcul du champ.

Exemples

Exemple d'extraction dans une liste de **RESULTAT** d'une composante du champ de déplacement en un noeud donné et pour une liste d'instants donnée, et récupération des valeurs dans un dictionnaire Python.

```
valeurs = {}
for ind_resu, resu in enumerate(liste_resu):
    valeurs[ind_resu] = []
    for ind_inst, inst in enumerate(list_inst):
        depl = resu.extraire(GRANDEUR = DEPLA_2D(V = 'OUI'),
                             LIEU = LIEU(NOMS = [nom_noeud],
                                         TYP = 'NOEUD'),
                             PAS = ind_inst + 1)
        valeurs[ind_resu].append( depl.val[nom_noeud]['V'] )
```

2.19.2 Classe DEPLA_2D

La classe **DEPLA_2D** est la classe permettant de définir les composantes du champ de déplacement bidimensionnel à extraire.

```
depl2d = DEPLA_2D(
        ♦ U = / 'OUI',
              / 'NON',
        ♦ V = / 'OUI',
              / 'NON',
        ) # fin DEPLA_2D
```

U : déplacement u dans le repère du maillage.
V : déplacement v dans le repère du maillage.

2.19.3 Classe DEPLA_3D

La classe **DEPLA_3D** est la classe permettant de définir les composantes du champ de déplacement tridimensionnel à extraire.

```
depl3d = DEPLA_3D(
        ♦ U = / 'OUI',
              / 'NON',
        ♦ V = / 'OUI',
              / 'NON',
        ♦ W = / 'OUI',
              / 'NON',
        ) # fin DEPLA_3D
```

U : déplacement u dans le repère du maillage.
V : déplacement v dans le repère du maillage.
W : déplacement w dans le repère du maillage.

2.19.4 Classe DEPLA_2D_ROTA

La classe **DEPLA_2D_ROTA** est la classe permettant de définir les composantes du champ de déplacement bidimensionnel généralisé (déplacement ou rotation) à extraire.

```
depl3drot = DEPLA_2D_ROTA(
          ♦ U = / 'OUI',
                / 'NON',
          ♦ V = / 'OUI',
                / 'NON',
          ♦ RZ = / 'OUI',
                 / 'NON',
          ) # fin DEPLA_2D_ROTA
```

U : déplacement u dans le repère du maillage.
V : déplacement v dans le repère du maillage.
RZ : rotation r_z dans le repère du maillage.

2.19.5 Classe DEPLA_3D_ROTA

La classe **DEPLA_3D_ROTA** est la classe permettant de définir les composantes du champ de déplacement tridimensionnel généralisé (déplacement ou rotation) à extraire.

```
depl3drot = DEPLA_3D_ROTA(
          ♦ U = / 'OUI',
                / 'NON',
          ♦ V = / 'OUI',
                / 'NON',
          ♦ W = / 'OUI',
                / 'NON',
          ♦ RX = / 'OUI',
                 / 'NON',
          ♦ RY = / 'OUI',
                 / 'NON',
          ♦ RZ = / 'OUI',
                 / 'NON',
          ) # fin DEPLA_3D_ROTA
```

U : déplacement u dans le repère du maillage.
V : déplacement v dans le repère du maillage.
W : déplacement w dans le repère du maillage.
RX : rotation r_x dans le repère du maillage.
RY : rotation r_y dans le repère du maillage.
RZ : rotation r_z dans le repère du maillage.

2.19.6 Classe ACCEL_2D

La classe **ACCEL_2D** est la classe permettant de définir les composantes du champ d'accélération bidimensionnel à extraire.

```
accel2d = ACCEL_2D(
        ♦ AX = / 'OUI',
               / 'NON',
        ♦ AY = / 'OUI',
               / 'NON',
        ) # fin ACCEL_2D
```

AX : accélération a_x dans le repère du maillage.
AY : accélération a_y dans le repère du maillage.

2.19.7 Classe ACCEL_3D

La classe **ACCEL_3D** est la classe permettant de définir les composantes du champ d'accélération tridimensionnel à extraire.

```
accel3d = ACCEL_3D(
        ♦ AX = / 'OUI',
```

```
                / 'NON',
♦ AY =          / 'OUI',
                / 'NON',
♦ AZ =          / 'OUI',
                / 'NON',
) # fin ACCEL_3D
```

AX : accélération a_x dans le repère du maillage.
AY : accélération a_y dans le repère du maillage.
AZ : accélération a_z dans le repère du maillage.

2.19.8 Classe CONTR_2D

La classe **CONTR_2D** est la classe permettant de définir les composantes du tenseur de contraintes planes à extraire.

```
contr2d = CONTR_2D(
          ♦ SXX =  / 'OUI',
                   / 'NON',
          ♦ SYY =  / 'OUI',
                   / 'NON',
          ♦ SXY =  / 'OUI',
                   / 'NON',
          ♦ SZZ =  / 'OUI',
                  / 'NON',
          ) # fin CONTR_2D
```

SXX : contrainte σ_{xx} dans le repère du maillage.
SYY : contrainte σ_{yy} dans le repère du maillage.
SXY : contrainte σ_{xy} dans le repère du maillage.
SZZ : contrainte σ_{zz} dans le repère du maillage.

2.19.9 Classe CONTR_3D

La classe **CONTR_3D** est la classe permettant de définir les composantes du tenseur de contraintes tridimensionnelles à extraire.

```
contr3d = CONTR_3D(
          ♦ SXX =  / 'OUI',
                   / 'NON',
          ♦ SYY =  / 'OUI',
                   / 'NON',
          ♦ SZZ =  / 'OUI',
                  / 'NON',
          ♦ SXY =  / 'OUI',
                   / 'NON',
          ♦ SYZ =  / 'OUI',
                   / 'NON',
```

♦ **SZX** = / **'OUI'**,
/ **'NON'**,
) # *fin CONTR_3D*

SXX : contrainte σ_{xx} dans le repère du maillage.
SYY : contrainte σ_{yy} dans le repère du maillage.
SZZ : contrainte σ_{zz} dans le repère du maillage.
SXY : contrainte σ_{xy} dans le repère du maillage.
SYZ : contrainte σ_{yz} dans le repère du maillage.
SZX : contrainte σ_{zx} dans le repère du maillage.

2.19.10 Classe DEFORM_TOTAL_2D

La classe **DEFORM_TOTAL_2D** est la classe permettant de définir les composantes du tenseur de déformations totales planes à extraire.

defortot2d = **DEFORM_TOTAL_2D**(
♦ **ETXX** = / **'OUI'**,
/ **'NON'**,
♦ **ETYY** = / **'OUI'**,
/ **'NON'**,
♦ **ETXY** = / **'OUI'**,
/ **'NON'**,
♦ **ETZZ** = / **'OUI'**,
/ **'NON'**,
) # *fin DEFORM_TOTAL_2D*

ETXX : déformation ϵ_{txx} dans le repère du maillage.
ETYY : déformation ϵ_{tyy} dans le repère du maillage.
ETXY : déformation ϵ_{txy} dans le repère du maillage.
ETZZ : déformation ϵ_{tzz} dans le repère du maillage.

2.19.11 Classe DEFORM_TOTAL_3D

La classe **DEFORM_TOTAL_3D** est la classe permettant de définir les composantes du tenseur de déformations totales tridimensionnelles à extraire.

defortot3d = **DEFORM_TOTAL_3D**(
♦ **ETXX** = / **'OUI'**,
/ **'NON'**,
♦ **ETYY** = / **'OUI'**,
/ **'NON'**,
♦ **ETZZ** = / **'OUI'**,
/ **'NON'**,
♦ **ETXY** = / **'OUI'**,
/ **'NON'**,
♦ **ETYZ** = / **'OUI'**,
/ **'NON'**,

♦ **ETZX** = / **'OUI'**,
/ **'NON'**,
) # *fin DEFORM_TOTAL_3D*

ETXX : déformation ϵtxx dans le repère du maillage.
ETYY : déformation ϵtyy dans le repère du maillage.
ETZZ : déformation ϵtzz dans le repère du maillage.
ETXY : déformation ϵtxy dans le repère du maillage.
ETYZ : déformation ϵtyz dans le repère du maillage.
ETZX : déformation ϵtzx dans le repère du maillage.

2.19.12 Classe REAC_2D

La classe **REAC_2D** est la classe permettant de définir les composantes des réactions bidimensionnelles à extraire.

reac2d = **REAC_2D**(
♦ **RFX** = / **'OUI'**,
/ **'NON'**,
♦ **RFY** = / **'OUI'**,
/ **'NON'**,
) # *fin REAC_2D*

RFX : réaction rf_x dans le repère du maillage.
RFY : réaction rf_y dans le repère du maillage.

2.19.13 Classe REAC_3D

La classe **REAC_3D** est la classe permettant de définir les composantes des réactions tridimensionnelles à extraire.

reac3d = **REAC_3D**(
♦ **RFX** = / **'OUI'**,
/ **'NON'**,
♦ **RFY** = / **'OUI'**,
/ **'NON'**,
♦ **RFZ** = / **'OUI'**,
/ **'NON'**,
) # *fin REAC_3D*

RFX : réaction rf_x dans le repère du maillage.
RFY : réaction rf_y dans le repère du maillage.
RFZ : réaction rf_z dans le repère du maillage.

2.19.14 Classe EFFORT_BAR_2D

La classe **EFFORT_BAR_2D** est la classe permettant de définir les composantes du champ d'efforts d'une barre bidimensionnelle à extraire.

effbar2d = **EFFORT_BAR_2D**(
♦ **N** = / **'OUI'**,
/ **'NON'**,
) # *fin EFFORT_BAR_2D*

N : effort normal n de direction x dans le repère local de la barre.

2.19.15 Classe EFFORT_BAR_3D

La classe **EFFORT_BAR_3D** est la classe permettant de définir les composantes du champ d'efforts d'une barre tridimensionnelle à extraire.

effbar3d = **EFFORT_BAR_3D**(
♦ **N** = / **'OUI'**,
/ **'NON'**,
) # *fin EFFORT_BAR_3D*

N : effort normal n de direction x dans le repère local de la barre.

2.19.16 Classe EFFORT_POUTR_2D

La classe **EFFORT_POUTR_2D** est la classe permettant de définir les composantes du champ d'efforts d'une poutre bidimensionnelle à extraire.

effpou2d = **EFFORT_POUTR_2D**(
♦ **N** = / **'OUI'**,
/ **'NON'**,
♦ **VY** = / **'OUI'**,
/ **'NON'**,
♦ **MZ** = / **'OUI'**,
/ **'NON'**,
) # *fin EFFORT_POUTR_2D*

N : effort normal n de direction x dans le repère local de la poutre.
VY : effort tranchant vy de direction y dans le repère local de la poutre.
MZ : moment m_z d'axe z dans le repère local de la poutre.

2.19.17 Classe EFFORT_POUTR_3D

La classe **EFFORT_POUTR_3D** est la classe permettant de définir les composantes du champ d'efforts d'une poutre tridimensionnelle à extraire.

```
effpou3d = EFFORT_POUTR_3D(
           ♦ N = / 'OUI',
                 / 'NON',
           ♦ VY = / 'OUI',
                  / 'NON',
           ♦ VZ = / 'OUI',
                  / 'NON',
           ♦ MX = / 'OUI',
                  / 'NON',
           ♦ MY = / 'OUI',
                  / 'NON',
           ♦ MZ = / 'OUI',
                  / 'NON',
           ) # fin EFFORT_POUTR_3D
```

N : effort normal n de direction x dans le repère local de la poutre.
VY : effort tranchant vy de direction y dans le repère local de la poutre.
VZ : effort tranchant vz de direction z dans le repère local de la poutre.
MX : moment m_x d'axe x dans le repère local de la poutre.
MY : moment m_y d'axe y dans le repère local de la poutre.
MZ : moment m_z d'axe z dans le repère local de la poutre.

2.19.18 Classe TEMP

La classe **TEMP** est la classe permettant de définir les composantes du champ de température à extraire.

```
temp = TEMP(
       ♦ T = / 'OUI',
             / 'NON',
       ) # fin TEMP
```

T : température T.

2.19.19 Classe GRAD_TEMP_2D

La classe **GRAD_TEMP_2D** est la classe permettant de définir les composantes du gradient bidimensionnel du champ de température à extraire.

```
gradt2d = GRAD_TEMP_2D(
          ♦ TX = / 'OUI',
                 / 'NON',
          ♦ TY = / 'OUI',
                 / 'NON',
          ) # fin GRAD_TEMP_2D
```

TX : composante dans la direction x du gradient de température dans le repère du maillage.
TY : composante dans la direction y du gradient de température dans le repère du maillage.

2.19.20 Classe GRAD_TEMP_3D

La classe **GRAD_TEMP_3D** est la classe permettant de définir les composantes du gradient tridimensionnel du champ de température à extraire.

gradt2d = **GRAD_TEMP_3D**(
♦ **TX** = / **'OUI'**,
/ **'NON'**,
♦ **TY** = / **'OUI'**,
/ **'NON'**,
♦ **TZ** = / **'OUI'**,
/ **'NON'**,
) # *fin GRAD_TEMP_3D*

TX : composante dans la direction x du gradient de température dans le repère du maillage.
TY : composante dans la direction y du gradient de température dans le repère du maillage.
TZ : composante dans la direction z du gradient de température dans le repère du maillage.

2.19.21 Classe FIELD_

La classe **FIELD_** est la classe dont relève les éléments de l'attribut **fields** (liste de **FIELD_**) de la classe **RESULTAT**. Elle fait partie de l'API Python de la librairie MED Mémoire dont on trouve la documentation dans le répertoire prérequis du répertoire d'installation du Pilote. Pour la consulter, ouvrir par exemple le fichier suivant avec un navigateur html :

— `prerequis/MED/share/doc/salome/gui/MED/medmem.html` pour MED Mémoire,
— `prerequis/MED/share/doc/salome/gui/MED/classMEDMEM_1_1FIELD__.html` pour la classe **FIELD_** en particulier.

2.20 Exemple d'étude

A titre d'exemple, on reproduit ci-dessous le script correspondant au cas test ssnp001a, qui illustre un calcul statique non linéaire pour un modèle plan en contrainte plane.
On commente ensuite les différentes parties de la mise en données.

2.20.1 Script

```
1  from modele_donnees import *
2
3  rep_trav = 'tests/atelier/'
4  fic_msh = rep_trav + 'ssnp001a.msh'
5  fic_pos = rep_trav + 'ssnp001a.pos'
6  fic_pos2 = rep_trav + 'ssnp001a2.pos'
7
8  mail = MAILLAGE(NOM = 'mon_mail',
9                  FIC_IN = FICHIER(NOM = fic_msh,
10                                  FORMAT = 'GMSH'),
11                 DIM = 2,
12                 DIM_ESPACE = 2)
13
14 mod = MODELE(NOM = 'mon_mod',
```

```
15                MAIL = mail,
16                AFFECT = [AFFECT(PROPRIETE = FORMUL_MECA(TYP = 'MECA_2D_CPLAN'),
17                                 LIEU = LIEU(NOMS = ['Surf'],
18                                             TYP = 'GROUP_MAILLE'))])
19
20  mat = MATERIAU(NOM = 'mon_mat',
21                 MAIL = mail,
22                 AFFECT = [AFFECT(PROPRIETE = VON_MISES_AVEC_ECROU(NOM = 'mon_comport',
23                                                                   RO = 25000.,
24                                                                   YOUNG = 11000000000.,
25                                                                   POISS = 0.2,
26                                                                   K = 6600000000.,
27                                                                   H = 5000000000.),
28                                   LIEU = LIEU(NOMS = ['Surf'],
29                                               TYP = 'GROUP_MAILLE'))])
30
31  cara = CARACTERISTIQUE(NOM = 'mon_cara',
32                         MODEL = mod,
33                         AFFECT = [AFFECT(PROPRIETE = CONTR_PLANE(NOM = 'mon_cplan',
34                                                                  EP = 0.1),
35                                          LIEU = LIEU(NOMS = ['Surf'],
36                                                      TYP = 'GROUP_MAILLE'))])
37
38  cond = COND_LIMITE(NOM = 'mon_cond',
39                     MODEL = mod,
40                     AFFECT = [AFFECT(DDL_IMPOSE(V = 0.),
41                                      LIEU(NOMS = ['Bord_1'],
42                                           TYP = 'GROUP_ARETE')),
43                               AFFECT(DDL_IMPOSE(U = 0.),
44                                      LIEU(NOMS = ['Bord_4'],
45                                           TYP = 'GROUP_ARETE'))])
46
47  cond2 = COND_LIMITE(NOM = 'mon_cond2',
48                      MODEL = mod,
49                      AFFECT = [AFFECT(DDL_IMPOSE(U = 0.,
50                                                  V = 0.),
51                                       LIEU(NOMS = ['Bord_4'],
52                                            TYP = 'GROUP_ARETE'))])
53
54  char = CHARGEMENT(NOM = 'mon_char',
55                    MODEL = mod,
56                    AFFECT = [AFFECT(PROPRIETE = FORC_ARETE_2D(FX = 10000000.),
57                                     LIEU = LIEU(NOMS = ['Bord_2'],
58                                                 TYP = 'GROUP_ARETE'))])
59
60  char2 = CHARGEMENT(NOM = 'mon_char2',
61                     MODEL = mod,
62                     AFFECT = [AFFECT(PROPRIETE = FORC_ARETE_2D(FY = -10000000.),
63                                      LIEU = LIEU(NOMS = ['Bord_3'],
64                                                  TYP = 'GROUP_ARETE'))])
65
66  cas = CAS_CHARGEMENT(NOM = 'mon_cas',
67                       CHARG = [char],
68                       COEF = [1.])
69
70  cas2 = CAS_CHARGEMENT(NOM = 'mon_cas2',
71                        CHARG = [char2],
72                        COEF = [1.])
73
74  ana = STAT_NON_LINE(NOM = 'mon_ana',
75                      SOLVEUR = 'MULTIFRONT',
76                      METH = METHOD_ITER(TYP = 'NEWTON_COMPLET',
77                                         NB_ITER = 10,
78                                         TOLER = 1e-4),
79                      INCREM = INCREMENTATION(NOM = 'increm',
80                                              LIST_INST = [1., 2., 3., 4., 5.]),
81                      FONC = [FONCT_MULT(NOM = 'ma_fonc',
82                                         LIST_INST = [1., 2., 3., 4., 5.],
83                                         LIST_COEF = [0.0, 0.25, 0.5, 0.75, 1.0],
84                                         NOM_CAS = 'mon_cas')])
85
86  ana2 = STAT_NON_LINE(NOM = 'mon_ana2',
87                       SOLVEUR = 'MULTIFRONT',
88                       METH = METHOD_ITER(TYP = 'NEWTON_COMPLET',
89                                          NB_ITER = 10,
```

```
                                          TOLER = 1e-4),
                        INCREM = INCREMENTATION(NOM = 'increm',
                                                LIST_INST = [1., 2., 3., 4., 5.]),
                        FONC = [FONCT_MULT(NOM = 'ma_fonc',
                                           LIST_INST = [1., 2., 3., 4., 5.],
                                           LIST_COEF = [0.0, 0.25, 0.5, 0.75, 1.0],
                                           NOM_CAS = 'mon_cas2')])

resol = RESOLUTION(NOM = 'ma_resol',
                   MODEL = mod,
                   MATER = mat,
                   CARAC = cara,
                   COND_LIM = cond,
                   CAS_CHARG = [cas],
                   ANALYS = ana)

resol2 = RESOLUTION(NOM = 'ma_resol2',
                    MODEL = mod,
                    MATER = mat,
                    CARAC = cara,
                    COND_LIM = cond2,
                    CAS_CHARG = [cas2],
                    ANALYS = ana2)

resu = RESULTAT(NOM = 'mon_resu',
                RESOL = resol,
                FIC_OUT = FICHIER(NOM = fic_pos,
                                  FORMAT = 'GMSH'))

resu2 = RESULTAT(NOM = 'mon_resu2',
                 RESOL = resol2,
                 FIC_OUT = FICHIER(NOM = fic_pos2,
                                   FORMAT = 'GMSH'))

etu = ETUDE(MAIL = [mail],
            RESOL = [resol, resol2],
            RESU = [resu, resu2],
            EXEC = PARAM_EXEC(TYP = 'DISTRIB',
                              NB_PROC = 2))

etu.lancer()
```

2.20.2 Commentaires

Analysons les différentes parties du script précédent.

Préambule. On commence par importer le langage Pilote :

```
from modele_donnees import *
```

puis on définit les noms des fichiers à gérer à l'aide des variables `fic_msh`, `fic_pos` et `fic_pos2`, en utilisant la variable `rep_trav` pour stocker le chemin d'accès commun à ces fichiers :

```
rep_trav = 'tests/atelier/'
fic_msh = rep_trav + 'ssnp001a.msh'
fic_pos = rep_trav + 'ssnp001a.pos'
fic_pos2 = rep_trav + 'ssnp001a2.pos'
```

Données de base. On définit ensuite l'ensembles des données relatives :

— au maillage :

```
mail = MAILLAGE(NOM = 'mon_mail',
                FIC_IN = FICHIER(NOM = fic_msh,
                                 FORMAT = 'GMSH'),
                DIM = 2,
                DIM_ESPACE = 2)
```

On importe un fichier de maillage au format GMSH dont le nom est spécifié dans `fic_msh` défini précédemment. Ce maillage est affecté du libellé `'mon_mail'` et on choisit de nommer `mail` l'instance de **MAILLAGE**.

— au modèle :

```
mod = MODELE(NOM = 'mon_mod',
             MAIL = mail,
             AFFECT = [AFFECT(PROPRIETE = FORMUL_MECA(TYP = 'MECA_2D_CPLAN'),
                              LIEU = LIEU(NOMS = ['Surf'],
                                          TYP = 'GROUP_MAILLE'))])
```

On affecte au groupe de mailles nommé `'Surf'` du maillage `mail` le modèle de mécanique bidimensionnelle en contrainte plane. L'ensemble des propriétés de modèle est affecté du libellé `'mon_mod'` et on choisit de nommer `mod` l'instance de **MODELE**.

— au matériau :

```
mat = MATERIAU(NOM = 'mon_mat',
               MAIL = mail,
               AFFECT = [AFFECT(PROPRIETE = VON_MISES_AVEC_ECROU(NOM = 'mon_comport',
                                                                 RO = 25000.,
                                                                 YOUNG = 11000000000.,
                                                                 POISS = 0.2,
                                                                 K = 6600000000.,
                                                                 H = 5000000000.),
                                LIEU = LIEU(NOMS = ['Surf'],
                                            TYP = 'GROUP_MAILLE'))])
```

On affecte au groupe de mailles nommé `'Surf'` du maillage `mail` la loi de comportement non linéaire de type Von-Mises avec écrouissage. Cette loi est affectée du libellé `'mon_comport'`. L'ensemble des propriétés de matériau est affecté du libellé `'mon_mat'` et on choisit de nommer `mat` l'instance de **MATERIAU**.

— aux caractéristiques :

```
cara = CARACTERISTIQUE(NOM = 'mon_cara',
                       MODEL = mod,
                       AFFECT = [AFFECT(PROPRIETE = CONTR_PLANE(NOM = 'mon_cplan',
                                                                EP = 0.1),
                                        LIEU = LIEU(NOMS = ['Surf'],
                                                    TYP = 'GROUP_MAILLE'))])
```

On affecte au groupe de mailles nommé `'Surf'` la caractéristique géométrique du modèle en contrainte plane, c'est à dire l'épaisseur de la structure. Cette caractéristique de contrainte plane est affectée du libellé `'mon_cplan'`. L'ensemble des propriétés de caractéristique est affecté du libellé `'mon_cara'` et s'applique au modèle `mod`. On choisit de nommer `cara` l'instance de **CARACTERISTIQUE**.

— aux conditions limites :

```
cond = COND_LIMITE(NOM = 'mon_cond',
                   MODEL = mod,
                   AFFECT = [AFFECT(DDL_IMPOSE(V = 0.),
                                    LIEU(NOMS = ['Bord_1'],
                                         TYP = 'GROUP_ARETE')),
                             AFFECT(DDL_IMPOSE(U = 0.),
                                    LIEU(NOMS = ['Bord_4'],
                                         TYP = 'GROUP_ARETE'))])
```

On affecte des conditions aux limites de type $v = 0$ sur le groupe d'arêtes nommé `'Bord_1'` et $u = 0$ sur le groupe d'arêtes nommé `'Bord_4'`. Ce premier ensemble de propriétés de conditions limites est affecté du libellé `'mon_cond'` et s'applique au modèle `mod`. On choisit de nommer `cond` l'instance de **COND_LIMITE**.

```
cond2 = COND_LIMITE(NOM = 'mon_cond2',
                    MODEL = mod,
                    AFFECT = [AFFECT(DDL_IMPOSE(U = 0.,
                                                V = 0.),
```

```
                                                    LIEU(NOMS = ['Bord_4'],
                                                         TYP = 'GROUP_ARETE'))])
```

On définit un deuxième jeu de conditions aux limites en affectant des conditions de type $u = 0$ et $v = 0$ sur le groupe d'arêtes nommé `'Bord_4'`. Ce deuxième ensemble de propriétés de conditions limites est affecté du libellé `'mon_cond2'` et s'applique au modèle `mod`. On choisit de nommer `cond2` l'instance de **COND_LIMITE**.

— aux cas de chargements :

```
char = CHARGEMENT(NOM = 'mon_char',
                  MODEL = mod,
                  AFFECT = [AFFECT(PROPRIETE = FORC_ARETE_2D(FX = 10000000.),
                                   LIEU = LIEU(NOMS = ['Bord_2'],
                                               TYP = 'GROUP_ARETE'))])
```

On définit un chargement de type force linéique de composante f_x sur arête d'élément 2D à appliquer sur le groupe d'arêtes nommé `'Bord_2'`. Cet ensemble de propriétés de chargement est affecté du libellé `'mon_char'` et s'applique au modèle `mod`. On choisit de nommer `char` l'instance de **CHARGEMENT**.

```
char2 = CHARGEMENT(NOM = 'mon_char2',
                   MODEL = mod,
                   AFFECT = [AFFECT(PROPRIETE = FORC_ARETE_2D(FY = -10000000.),
                                    LIEU = LIEU(NOMS = ['Bord_3'],
                                                TYP = 'GROUP_ARETE'))])
```

On définit un deuxième chargement de type force linéique de composante f_y sur arête d'élément 2D à appliquer sur le groupe d'arêtes nommé `'Bord_3'`. Ce deuxième ensemble de propriétés de chargement est affecté du libellé `'mon_char2'` et s'applique au modèle `mod`. On choisit de nommer `char2` l'instance de **CHARGEMENT**.

```
cas = CAS_CHARGEMENT(NOM = 'mon_cas',
                     CHARG = [char],
                     COEF = [1.])

cas2 = CAS_CHARGEMENT(NOM = 'mon_cas2',
                      CHARG = [char2],
                      COEF = [1.])
```

A partir des chargements `char` et `char2` précédemment définis, on définit deux cas de chargement. Le premier correspond trivialement au chargement `char` (affecté du coefficient 1) et est affecté du libellé `'mon_cas'` Le deuxième correspond au chargement `char2` et est affecté du libellé `'mon_cas2'` On choisit de nommer respectivement `cas` et `cas2` les 2 instances de **CAS_CHARGEMENT**.

— aux analyses :

```
ana = STAT_NON_LINE(NOM = 'mon_ana',
                    SOLVEUR = 'MULTIFRONT',
                    METH = METHOD_ITER(TYP = 'NEWTON_COMPLET',
                                       NB_ITER = 10,
                                       TOLER = 1e-4),
                    INCREM = INCREMENTATION(NOM = 'increm',
                                            LIST_INST = [1., 2., 3., 4., 5.]),
                    FONC = [FONCT_MULT(NOM = 'ma_fonc',
                                       LIST_INST = [1., 2., 3., 4., 5.],
                                       LIST_COEF = [0.0, 0.25, 0.5, 0.75, 1.0],
                                       NOM_CAS = 'mon_cas')])
```

On définit une analyse statique non linéaire qui fera appel au solveur multifrontal et qui mettra en oeuvre une méthode itérative de Newton complète. Le processus d'incrémentation de charge est défini d'une part par une liste de pas de pseudo-temps (affectée du libellé `'increm'`) et d'autre part par une fonction multiplicatrice à appliquer au cas de chargement de nom `mon_cas` (affectée du libellé `'ma_fonc'`). Cette analyse est affectée du libellé `'mon_ana'` et on choisit de nommer `ana` l'instance de **STAT_NON_LINE**.

```
ana2 = STAT_NON_LINE(NOM = 'mon_ana2',
                     SOLVEUR = 'MULTIFRONT',
                     METH = METHOD_ITER(TYP = 'NEWTON_COMPLET',
                                        NB_ITER = 10,
                                        TOLER = 1e-4),
                     INCREM = INCREMENTATION(NOM = 'increm',
                                             LIST_INST = [1., 2., 3., 4., 5.]),
                     FONC = [FONCT_MULT(NOM = 'ma_fonc',
                                        LIST_INST = [1., 2., 3., 4., 5.],
                                        LIST_COEF = [0.0, 0.25, 0.5, 0.75, 1.0],
                                        NOM_CAS = 'mon_cas2')])
```

On définit une deuxième analyse analogue à la précédente, mais dont la fonction multiplicatrice s'applique au cas de chargement de nom `mon_cas2`. Cette analyse est affectée du libellé `'mon_ana2'` et on choisit de nommer `ana2` l'instance de **STAT_NON_LINE**.

Résolutions. Après avoir défini les blocs de données de base, on définit les cas d'étude à résoudre par le solveur.
On définit un premier cas d'étude (ou résolution) en invoquant des instances d'objets précédemment construits. En particulier, on fait référence ici au jeu de conditions limites `cond`, au cas de chargement `cas` et à l'analyse `ana`. Cette résolution est affectée du libellé `'ma_resol'` et on choisit de nommer `resol` l'instance de **RESOLUTION**.

```
resol = RESOLUTION(NOM = 'ma_resol',
                   MODEL = mod,
                   MATER = mat,
                   CARAC = cara,
                   COND_LIM = cond,
                   CAS_CHARG = [cas],
                   ANALYS = ana)
```

De façon analogue, on définit ensuite une deuxième résolution référençant `cond2`, `cas2` et `ana2`. Elle est affectée du libellé `'ma_resol2'` et on choisit de nommer `resol2` l'instance de **RESOLUTION**.

```
resol2 = RESOLUTION(NOM = 'ma_resol2',
                    MODEL = mod,
                    MATER = mat,
                    CARAC = cara,
                    COND_LIM = cond2,
                    CAS_CHARG = [cas2],
                    ANALYS = ana2)
```

Résultats. Après résolution, on souhaite récupérer les résultats pour les visualiser. On demande l'exportation d'un premier fichier de résultats, associé à la résolution `resol`, au format GMSH et dont le nom est spécifié dans `fic_pos` défini en début de script. Ce résultat est affecté du libellé `'mon_resu'` et on choisit de nommer `resu` l'instance de **RESULTAT** :

```
resu = RESULTAT(NOM = 'mon_resu',
                RESOL = resol,
                FIC_OUT = FICHIER(NOM = fic_pos,
                                  FORMAT = 'GMSH'))
```

De façon analogue, on définit un résultat associé à la résolution `resol2`. Ce résultat est affecté du libellé `'mon_resu2'` et on choisit de nommer `resu2` l'instance de **RESULTAT** :

```
resu2 = RESULTAT(NOM = 'mon_resu2',
                 RESOL = resol2,
                 FIC_OUT = FICHIER(NOM = fic_pos2,
                                   FORMAT = 'GMSH'))
```

Etude. On termine le script en définissant l'étude à exécuter en faisant référence :

- aux maillages qu'elle doit importer (`mail`),
- aux cas d'étude qu'elle doit résoudre (`resol` et `resol2`),
- aux résultats qu'elle doit exporter (`resu` et `resu2`),

et en définissant ses paramètres d'exécution (ici on distribue les 2 résolutions sur 2 processeurs). On choisit de nommer `etu` l'instance de **ETUDE** :

```
etu = ETUDE(MAIL = [mail],
            RESOL = [resol, resol2],
            RESU = [resu, resu2],
            EXEC = PARAM_EXEC(TYP = 'DISTRIB',
                              NB_PROC = 2))
```

Il reste à demander l'exécution de l'étude via sa méthode **lancer** :

```
etu.lancer()
```

Chapitre 3

Langage de commandes CESAR

Le langage CESAR permet de réaliser un calcul avec le solveur CESAR, c'est à dire de générer son fichier de données (.data) et de lancer son exécution au moyen d'un script Python utilisant les commandes du langage.
Les commandes du langage sont "adhérentes" à celles utilisées pour le fichier de données (.data) de CESAR, d'où le nom donné à ce langage. Ce langage permet ainsi de retrouver la mise en données "classique" du solveur CESAR (donc avec le même bas niveau d'abstraction) mais dans un environnement Python. Il améliore donc la lisibilité de la mise en données tout en fournissant à l'utilisateur le confort de l'environnement Python pour définir des paramètres, réaliser du scripting, ...
Contrairement au langage Pilote, et compte-tenu de son bas niveau d'abstraction, le langage CESAR ne permet pas réellement d'interfacer le solveur avec des outils de maillage et de visualisation de résultats. Néanmoins, 2 fonctionnalités peuvent être utilisées :

- d'une part, le solveur est désormais équipé d'un nouveau module utilitaire (module EXPO) permettant l'export du maillage et des résultats du calcul au format GMSH, et ce module peut donc naturellement être invoqué via le langage CESAR (voir la description de la classe **EXPO** à la section 3.3.5),
- d'autre part, le langage CESAR introduit une fonctionnalité (classe **MAIL**), qui n'existe pas dans l'interface native du solveur, permettant de lire un maillage au format GMSH moyennant quelques limitations (voir la description de **MAIL** à la section 3.30.1).

On décrit dans ce chapitre la syntaxe du langage CESAR, dont les mots-clé sont issus pour l'essentiel de l'interface native de CESAR. Pour être autoporteur, on rappelle la signification de ces mots-clé mais de façon très synthétique. En cas de besoin, il convient donc de se reporter au "Manuel de référence" de CESAR.

Le langage CESAR peut être utilisé de façon directe (interface textuelle) ou indirecte (interface graphique).

Interface textuelle. La mise en données d'une étude (jeu de données) s'effectue en écrivant un script Python (par exemple `jdd.py`). Du point de vue Python, le langage CESAR est vu comme un "module" que l'on doit importer en tête du script Python en écrivant la ligne :

```
from modele_cesar import *
```

L'exécution de l'étude se fait en appelant (dans un terminal) le script `run_pilote` (présent dans le répertoire `bin/` de la distribution du Pilote) et en lui passant le script des données :

```
> run_pilote jdd.py
```

Interface graphique. La mise en donnée de l'étude peut également s'effectuer de façon graphique en appelant le script `ihm_cesar` (présent dans le répertoire `bin/` de la distribution du Pilote) qui lance un outil intéractif permettant de saisir et de visualiser graphiquement (sous forme d'arbre) l'ensemble des données d'une étude, ainsi que de générer automatiquement le script Python correspondant aux données saisies.

Exemple A titre d'exemple, voici comment lancer le cas test ssnp007 présent dans le répertoire `pilote/tests/donnees`. Ci-dessous, `cespil` désigne le répertoire d'installation du Pilote et `home` désigne le répertoire de l'utilisateur.
De façon préalable :

- on crée un répertoire de travail, par exemple du nom du test : `home/ssnp007`
- on y copie les fichiers en entrée du test : script `ssnp007.py` (pas de fichier de maillage, le maillage étant défini dans le script)
- on édite le script et on met à jour la variable `rep_trav` : `rep_trav = 'home/ssnp007/'`

Pour lancer l'exécution, on utilise le script `run_pilote` :

- on ouvre un terminal et on se place dans le répertoire de travail : `cd home/ssnp007`
- on appelle le script : `cespil/bin/run_pilote ssnp007.py`

De façon alternative, on peut utiliser le script `ihm_cesar` :

- on lance l'IHM : `cespil/bin/ihm_cesar`
- on lance l'exécution simplement en important le script avec l'explorateur : `File` / `Import` / `ssnp007.py`

3.1 Principe général de mise en données

L'idée principale consiste à instancier et renseigner un modèle de données à partir des commandes du langage.
On doit ainsi définir un jeu de données (classe **JDD**) qu'on peut ensuite **lancer**, c'est à dire exécuter à l'aide du solveur CESAR.
Un jeu de données doit comporter l'ensemble des données de type :

- données utilitaires (choix du type de passage en mode exécution ou test, ...),
- données de modèle (coordonnées, propriétés élémentaires, conditions limites, chargements),
- données de calcul (type d'analyse : linéaire, non linéaire, ...)

En pratique, on commence donc par définir des ensembles de données de ce type à l'aide des classes :

- **EXEC**, **TEST**, ... correspondant aux modules EXEC, TEST, ... du solveur,
- **COOR** correspondant au module COOR du solveur,
- **ELEM** correspondant au module ELEM du solveur,
- **LINE**, **MCNL**, ... correspondant aux modules d'exécution LINE, MCNL, ... du solveur.

Puis on peut alors construire un jeu de données à l'aide de la classe **JDD** en invoquant des références aux objets précédents construits.
On lance l'exécution du jeu de données à l'aide la méthode **lancer** de la classe **JDD**.

3.2 Définition d'un jeu de données

3.2.1 Classe JDD

La classe **JDD** permet de définir un jeu de données pour un calcul CESAR.

jdd = **JDD**(
- ♦ / **TEST** = test, [**TEST**]
  / **EXEC** = exec, [**EXEC**]
- ◊ **ILIG** = ilig, [**ILIG**]
- ◊ **COMT** = comt, [**COMT**]
- ◊ **EXPO** = expo, [**EXPO**]
- ♦ **COOR** = coor, [**COOR**]
- ♦ **ELEM** = elem, [**ELEM**]
- ♦ | **COND** = cond, [**list**<**COND**>]
  | **CHAR** = char, [**list**<**CHAR**>]
- ◊ **GEFI** = gefi, [**GEFI**]
- ◊ **IMPR** = impr, [**IMPR**]
- ♦ / **LINE** = line, [**LINE**]
  / **MCNL** = mcnl, [**MCNL**]
  / **FLAM** = flam, [**FLAM**]
  / **MODE** = mode, [**MODE**]
  / **LINH** = linh, [**LINH**]
  / **LINC** = linc, [**LINC**]
  / **DYNI** = dyni, [**DYNI**]
  / **SUMO** = sumo, [**SUMO**]
  / **TCNL** = tcnl, [**TCNL**]
  / **DTNL** = dtnl, [**DTNL**]
  / **DTLI** = dtli, [**DTLI**]
  / **NSAT** = nsat, [**NSAT**]
  / **TEXO** = texo, [**TEXO**]
  / **MEXO** = mexo, [**MEXO**]
  / **MPNL** = mpnl, [**MPNL**]
  / **MPLI** = mpli, [**MPLI**]
  / **AXIF** = axif, [**AXIF**]
  / **SURF** = surf, [**SURF**]

) # *fin JDD*

TEST : objet de type **TEST** associé au module TEST du solveur.
EXEC : objet de type **EXEC** associé au module EXEC du solveur.
ILIG : objet de type **ILIG** associé au module ILIG du solveur.
COMT : objet de type **COMT** associé au module COMT du solveur.
EXPO : objet de type **EXPO** associé au module EXPO du solveur.
COOR : objet de type **COOR** associé au module COOR du solveur.
ELEM : objet de type **ELEM** associé au module ELEM du solveur.
COND : objet de type **COND** associé au module COND du solveur.
CHAR : objet de type **CHAR** associé au module CHAR du solveur.
GEFI : objet de type **GEFI** associé au module GEFI du solveur.
IMPR : objet de type **IMPR** associé au module IMPR du solveur.
LINE : objet de type **LINE** associé au module LINE du solveur.
MCNL : objet de type **MCNL** associé au module MCNL du solveur.
FLAM : objet de type **FLAM** associé au module FLAM du solveur.
MODE : objet de type **MODE** associé au module MODE du solveur.
LINH : objet de type **LINH** associé au module LINH du solveur.
LINC : objet de type **LINC** associé au module LINC du solveur.

DYNI : objet de type **DYNI** associé au module DYNI du solveur.
SUMO : objet de type **SUMO** associé au module SUMO du solveur.
TCNL : objet de type **TCNL** associé au module TCNL du solveur.
DTNL : objet de type **DTNL** associé au module DTNL du solveur.
DTLI : objet de type **DTLI** associé au module DTLI du solveur.
NSAT : objet de type **NSAT** associé au module NSAT du solveur.
TEXO : objet de type **TEXO** associé au module TEXO du solveur.
MEXO : objet de type **MEXO** associé au module MEXO du solveur.
MPNL : objet de type **MPNL** associé au module MPNL du solveur.
MPLI : objet de type **MPLI** associé au module MPLI du solveur.
AXIF : objet de type **AXIF** associé au module AXIF du solveur.
SURF : objet de type **SURF** associé au module SURF du solveur.

3.3 Données utilitaires

3.3.1 Classe TEST

La classe **TEST** permet de réaliser un passage CESAR en mode test (vérification de cohérence des données du calcul). Il permet de définir les noms des différents fichiers impliqués dans l'appel du solveur.

test = **TEST**(
◊ **DATA** = data, [**str**]
◊ **MAIL_RESU** = mail_resu, [**str**]
◊ **RSV4** = rsv4, [**str**]
◊ **MESG** = mesg, [**str**]
◊ **LIST** = list, [**str**]
◊ **MSH** = msh, [**str**]
◊ **CFOR** = cfor, [**str**]
◊ **INI** = ini, [**list**<**str**>]
◊ **STK** = stk, [**list**<**str**>]
◊ **PRTX** = stk, [**list**<**str**>]
) # *fin TEST*

DATA : nom du fichier de données (.data).
MAIL_RESU : nom du fichier binaire de maillage généré par le solveur (_mail.resu).
RSV4 : nom du fichier binaire de résultats généré par le solveur (.rsv4).
MESG : nom du fichier de message généré par le solveur (.mesg).
LIST : nom du fichier de listage des résultats généré par le solveur (.list).
MSH : nom du fichier de maillage et de résultats généré par le solveur au format GMSH (.msh).
CFOR : nom du fichier de résultat généré par le solveur (.cfor).
INI : nom du fichier d'initialisation (.ini).
STK : nom du fichier de stockage de résultats créé par l'option de calcul STK (.stk).
PRTX : nom du fichier de stockage de résultats créé par l'option de calcul PTX (.prtx).

3.3.2 Classe EXEC

La classe **EXEC** permet de réaliser un passage CESAR en mode exécution (exécution effective du calcul).

exec = **EXEC**(
◊ **DATA** = data, [**str**]
◊ **MAIL_RESU** = mail_resu, [**str**]
◊ **RSV4** = rsv4, [**str**]
◊ **MESG** = mesg, [**str**]
◊ **LIST** = list, [**str**]
◊ **MSH** = msh, [**str**]
◊ **CFOR** = cfor, [**str**]
◊ **INI** = ini, [**list**<**str**>]
◊ **STK** = stk, [**list**<**str**>]
◊ **PRTX** = stk, [**list**<**str**>]
) # *fin EXEC*

DATA : nom du fichier de données (.data).
MAIL_RESU : nom du fichier binaire de maillage généré par le solveur (_mail.resu).
RSV4 : nom du fichier binaire de résultats généré par le solveur (.rsv4).
MESG : nom du fichier de message généré par le solveur (.mesg).
LIST : nom du fichier de listage des résultats généré par le solveur (.list).
MSH : nom du fichier de maillage et de résultats généré par le solveur au format GMSH (.msh).
CFOR : nom du fichier de résultat généré par le solveur (.cfor).
INI : nom du fichier d'initialisation (.ini).
STK : nom du fichier de stockage de résultats créé par l'option de calcul STK (.stk).
PRTX : nom du fichier de stockage de résultats créé par l'option de calcul PTX (.prtx).

3.3.3 Classe ILIG

La classe **ILIG** permet d'imprimer l'ensemble des lignes du jeu de données dans le fichier de listage des résultats.

ilig = **ILIG**() # *fin ILIG*

3.3.4 Classe COMT

La classe **COMT** permet d'imprimer des commentaires dans le fichier de listage des résultats.

comt = **COMT**(
♦ **LIGNES** = lignes, [**list**<**str**>]
) # *fin COMT*

LIGNES : liste de chaines de caractères utilisées pour les commentaires.

3.3.5 Classe EXPO

La classe **EXPO** permet d'exporter le maillage et les résultats du calcul dans un fichier au format spécifié. Dans la version actuelle, seul le format GMSH est disponible.

```
expo = EXPO(
       ♦ IPGMSH = / 0,
                  / 1,
       ) # fin EXPO
```

IPGMSH : entier indiquant la création (**1**) ou non (**0**) du fichier .msh au format GMSH.

3.3.6 Classe GEFI

La classe **GEFI** permet la gestion ou le contrôle du fichier de résultats.

```
gefi = GEFI(
       ♦ IPRESU = / 0,
                  / 1,
       ♦ IVERFC = / 0,
                  / 1,
                  / 2,
                  / 3,
       ) # fin GEFI
```

IPRESU : entier indiquant la création (**1**) ou non (**0**) du fichier de résultats .resu.
IVERFC : entier indiquant le niveau de vérification du fichier de résultats .rsv4 à choisir parmi :
— **0** : pas de vérification,
— **1** : création du fichier .cfor,
— **2** : écriture dans le fichier .list des en-têtes présents dans le .rsv4,
— **3** : création du .cfor + en-têtes dans le .list.

3.3.7 Classe IMPR

La classe **IMPR** permet de définir les impressions souhaitées dans le fichier dse listage des résultats.

```
impr = IMPR(
       ♦ IIP = / 0,
               / 1,
                 ♦ GENP = genp, [list <| GEN_IIP # inconnues principales
                                      >] # fin list
               / 2,
       ♦ IRC = / 0,
               / 1,
                 ♦ GENC = genp, [list <| GEN_IRC # résultats complémentaires
                                      >] # fin list
               / 2,
       ) # fin IMPR
```

IIP : entier indiquant le niveau d'impression des inconnues principales à choisir parmi **0**, **1**, **2**.
GENP : liste d'objets de type **GEN_IIP** définissant les impressions des inconnues principales.
IRC : entier indiquant le niveau d'impression des résultats complémentaires à choisir parmi **0**, **1**, **2**.
GENC : liste d'objets de type **GEN_IRC** définissant les impressions des résultats complémentaires.

Inconnues principales. La classe **GEN_IIP** permet de définir l'impression des résultats sur les inconnues principales.

```
gen_iip = GEN_IIP(
        ♦ IGENP = / 1,
                    ♦ ID = id, [int]
                    ♦ IF = if, [int]
                    ♦ IPAS = ipas, [int]
                  / 2,
                    ♦ NN = nn, [int]
                    ♦ NUM = num, [list<int>]
                  / 3,
                    ♦ NG = ng, [int]
                    ♦ NUG = nug, [list<int>]
        ) # fin GEN_IIP
```

IGENP : entier indiquant le mode de génération des noeuds :
— **1** : pour une génération par **ID**, **IF** et **IPAS**,
— **2** : pour une génération par **NN** et **NUM**,
— **3** : pour une génération par **NG** et **NUG**.
ID : entier donnant le premier numéro à générer.
IF : entier donnant le dernier numéro à générer.
IPAS : entier donnant le pas de progression entre **ID** et **IF**.
NN : entier donnant le nombre de numéros de noeuds à définir.
NUM : liste d'entiers donnant les **NN** numéros des noeuds.
NG : entier donnant le nombre de numéros de groupes d'éléments à définir.
NUG : liste d'entiers donnant les **NG** numéros des groupes d'éléments.

Résultats complémentaires. La classe **GEN_IRC** permet de définir l'impression des résultats complémentaires.

```
gen_irc = GEN_IRC(
        ♦ IGENC = / 1,
                    ♦ ID = id, [int]
                    ♦ IF = if, [int]
                    ♦ IPAS = ipas, [int]
                  / 2,
                    ♦ NE = nn, [int]
                    ♦ NUE = num, [list<int>]
                  / 3,
```

```
                    ♦ NG = ng, [int]
                    ♦ NUG = nug, [list<int>]
    ) # fin GEN_IIC
```

IGENC : entier indiquant le mode de génération des éléments :
— **1** : pour une génération par **ID**, **IF** et **IPAS**,
— **2** : pour une génération par **NN** et **NUM**,
— **3** : pour une génération par **NG** et **NUG**.
ID : entier donnant le premier numéro à générer.
IF : entier donnant le dernier numéro à générer.
IPAS : entier donnant le pas de progression entre **ID** et **IF**.
NE : entier donnant le nombre de numéros d'éléments à définir.
NUE : liste d'entiers donnant les **NE** numéros d'éléments.
NG : entier donnant le nombre de numéros de groupes d'éléments à définir.
NUG : liste d'entiers donnant les **NG** numéros des groupes d'éléments.

3.4 Données relatives aux noeuds

3.4.1 Classe COOR

La classe **COOR** permet de définir les coordonnées des noeuds du maillage.

```
coor = COOR(
        ♦ M = / 0,
              / 1,
              / 2,
        ♦ M1 = / 0,
                 ♦ NNT = nnt, [int]
                 ♦ NDIM = / 2,
                          / 3,
                 ♦ VCORG = vcorg, [list<float>]
               / 1,
    ) # fin COOR
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
M1 : entier indiquant l'utilisation (**1**) ou non (**0**) du fichier de maillage binaire.
NNT : entier donnant la valeur du nombre de noeuds total.
NDIM : entier donnant la valeur de la dimension du problème.
VCORG : liste de réels donnant les coordonnées des noeuds.

3.5 Données relatives aux éléments

3.5.1 Classe ELEM

La classe **ELEM** permet de définir les caractéristiques des éléments du maillage.

```
elem = ELEM(
```

♦ **M** = / **0**,
/ **1**,
/ **2**,
♦ **M1** = / **0**,
♦ **NELT** = nelt, [**int**]
♦ **NGRPE** = ngrpe, [**int**]
♦ **PNUMEL** = pnumel, [**list**<**int**>]
♦ **NUMEL** = numel, [**list**<**int**>]
♦ **TYPE** = type, [**list**<**str**>]
♦ **GROUPE** = groupe, [**list**<**int**>]
/ **1**,
♦ **GRPES** = grpes, [**list** <| **MB**
| **MT**
| **CO**
| **PB**
| **PT**
| **BB**
| **BT**
| **DB**
| **DT**
| **EB**
| **ET**
| **SB**
| **OB**
| **OT**
| **AX**
| **FD**
| **EJ**
| **RL**
| **SP**
>] # *fin list*
) # *fin ELEM*

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
M1 : entier indiquant l'utilisation (**1**) ou non (**0**) du fichier de maillage binaire.
NELT : entier donnant le nombre d'éléments total.
NGRPE : entier donnant le nombre de groupes d'éléments.
PNUMEL : liste d'entiers donnant les pointeurs indiquant le début des éléments dans **NUMEL**.
NUMEL : liste d'entiers donnant les connectivités nodales des éléments.
TYPE : liste de chaînes de caractères donnant les types des éléments.
GROUPE : liste d'entiers donnant les groupes d'appartenance des éléments.
GRPES : liste d'objets de type à choisir parmi :

- **MB** pour des éléments de mécanique bidimensionnelle,
- **MT** pour des éléments de mécanique tridimensionnelle,
- **CO** pour des éléments de coque (standard ou multicouche),
- **PB** pour des éléments de poutre bidimensionnelle,
- **PT** pour des éléments de poutre tridimensionnelle (standard ou multifibre),
- **BB** pour des éléments de barre bidimensionnelle,

- **BT** pour des éléments de barre tridimensionnelle,
- **DB** pour des éléments de diffusion bidimensionnelle,
- **DT** pour des éléments de diffusion tridimensionnelle,
- **EB** pour des éléments d'échange bidimensionnel,
- **ET** pour des éléments d'échange tridimensionnel,
- **SB** pour des éléments bidimensionnels pour recherche de surface libre,
- **OB** pour des éléments de thermo-mécanique bidimensionnelle des milieux poreux,
- **OT** pour des éléments de thermo-mécanique tridimensionnelle des milieux poreux,
- **AX** pour des éléments de mécanique bidimensionnelle axisymétrique,
- **FD** pour des éléments de frottement décollement (contact),
- **EJ** pour des éléments de joint,
- **RL** pour des éléments de relation linéaire,
- **SP** pour des éléments spéciaux (rigidité, masse, amortissement, ...).

3.6 Données relatives au modèle de mécanique bidimensionnelle

3.6.1 Classe MB

La classe **MB** permet de définir les caractéristiques d'un groupe d'éléments de mécanique bidimensionnelle.

```
mb = MB(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                 ♦ IMOD = / 1,
                            ♦ CARA = cara, [MB_ELI]
                          / 2,
                            ♦ CARA = cara, [MB_ELO]
                          / 5,
                            ♦ CARA = cara, [MB_BJA]
                          / 10,
                            ♦ CARA = cara, [MB_MCSE]
                          / 11,
                            ♦ CARA = cara, [MB_VMSE]
                          / 12,
                            ♦ CARA = cara, [MB_VMAE]
                          / 13,
                            ♦ CARA = cara, [MB_DPSE]
                          / 14,
                            ♦ CARA = cara, [MB_DPAE]
                          / 15,
                            ♦ CARA = cara, [MB_CP]
                          / 16,
                            ♦ CARA = cara, [MB_VE]
                          / 17,
                            ♦ CARA = cara, [MB_NO]
                          / 18,
                            ♦ CARA = cara, [MB_CCM]
```

```
                    / 19,
                      ♦ CARA = cara, [MB_PH]
                    / 20,
                      ♦ CARA = cara, [MB_CO]
                    / 24,
                      ♦ CARA = cara, [MB_HB]
                    / 34,
                      ♦ CARA = cara, [MB_ME]
                    / 37,
                      ♦ CARA = cara, [MB_MCSE_EO]
                    / 40,
                      ♦ CARA = cara, [MB_TA]
                    / 43,
                      ♦ CARA = cara, [MB_RBS]
                    / 47,
                      ♦ CARA = cara, [MB_WWS]
                    / 48,
                      ♦ CARA = cara, [MB_WWM]
                    / 88,
                      ♦ CARA = cara, [MB_EDI]
                    / 10000,
                      ♦ CARA = cara, [MB_BAO]
          / 'I',
) # fin MB
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **5**, **10**, **11**, **12**, **13**, **14**, **15**, **16**, **17**, **18**, **19**, **20**, **24**, **34**, **37**, **40**, **43**, **47**, **48**, **88**.
CARA : objet de type à choisir parmi :

- **MB_ELI** pour une loi de type élasticite lineaire isotrope,
- **MB_ELO** pour une loi de type élasticite lineaire orthotrope,
- **MB_BJA** pour une loi de type béton au jeune âge,
- **MB_MCSE** pour une loi de type Mohr-Coulomb sans écrouissage,
- **MB_VMSE** pour une loi de type Von Mises sans écrouissage,
- **MB_VMAE** pour une loi de type Von Mises avec écrouissage,
- **MB_DPSE** pour une loi de type Drucker-Prager sans écrouissage,
- **MB_DPAE** pour une loi de type Drucker-Prager avec écrouissage,
- **MB_CP** pour une loi de type critère parabolique,
- **MB_VE** pour une loi de type Vermeer,
- **MB_NO** pour une loi de type Nova,
- **MB_CCM** pour une loi de type Cam Clay modifié,
- **MB_PH** pour une loi de type Prevost-Hoeg,
- **MB_CO** pour une loi de type critère orienté,
- **MB_HB** pour une loi de type Hoek Brown,
- **MB_ME** pour une loi de type Mélanie,
- **MB_MCSE_EO** pour une loi de type Mohr Coulomb orthotrope,
- **MB_TA** pour une loi de type Tresca anisotrope,
- **MB_RBS** pour une loi de type matériau renforcé (Buhan et Dudret),

· **MB_WWS** pour une loi de type William-Warnke standard,
· **MB_WWM** pour une loi de type William-Warnke modifié,
· **MB_EDI** pour une loi de type élasticite avec dilatance isotrope,
· **MB_BAO** pour un modèle à composantes,
en cohérence avec le choix effectué pour **IMOD**.

3.6.2 Classe MB_ELI

La classe **MB_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de mécanique bidimensionnelle.

```
eli = MB_ELI(
    ♦ RO = ro, [float]
    ♦ YOUNG = young, [float]
    ♦ POISS = poiss, [float]
    ♦ INAT = / 1,
             / 2,
             / 3,
               ♦ EP = ep, [float]
    ) # fin MB_ELI
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.3 Classe MB_ELO

La classe **MB_ELO** permet d'affecter une loi de comportement de type élasticité linéaire orthotrope aux éléments de mécanique bidimensionnelle.

```
elo = MB_ELO(
    ♦ RO = ro, [float]
    ♦ E1 = e1, [float]
    ♦ E2 = e2, [float]
    ♦ P1 = p1, [float]
    ♦ P2 = p2, [float]
    ♦ G2 = g2, [float]
    ♦ TETA = teta, [float]
    ♦ INAT = / 1,
             / 2,
             / 3,
    ) # fin MB_ELO
```

RO : réel donnant la valeur de la masse volumique.

E1 : réel donnant la valeur du module d'Young dans la direction 1.
E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant l'angle entre l'axe Ox et la direction 1.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.

3.6.4 Classe MB_BJA

La classe **MB_BJA** permet d'affecter une loi de comportement de type béton au jeune âge aux éléments de mécanique bidimensionnelle.

```
bja = MB_BJA(
    ♦ RO = ro, [float]
    ♦ YOUNG = young, [float]
    ♦ POISS = poiss, [float]
    ♦ DILAT = dilat, [float]
    ♦ RETRA = retra, [float]
    ♦ SEUIL = seuil, [float]
    ♦ HYD = hyd, [float]
    ♦ INAT = / 1,
             / 2,
             / 3,
               ♦ EP = ep, [float]
    ) # fin MB_BJA
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
DILAT : réel donnant la valeur du coefficient de dilatation thermique.
RETRA : réel donnant la valeur du retrait endogène final.
SEUIL : réel donnant la valeur du seuil du matériau durci.
HYD : réel donnant la valeur du degré d'hydratation.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.5 Classe MB_MCSE

La classe **MB_MCSE** permet d'affecter une loi de comportement de type Mohr-Coulomb sans écrouissage aux éléments de mécanique bidimensionnelle.

```
mcse = MB_MCSE(
```

♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin MB_MCSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.6 Classe MB_VMSE

La classe **MB_VMSE** permet d'affecter une loi de comportement de type Von Mises sans écrouissage aux éléments de mécanique bidimensionnelle.

vmse = **MB_VMSE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **K** = k, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin MB_VMSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en ciaillement simple.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.7 Classe MB_VMAE

La classe **MB_VMAE** permet d'affecter une loi de comportement de type Von Mises avec écrouissage aux éléments de mécanique bidimensionnelle.

vmae = **MB_VMAE**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **K** = k, [**float**]
- ♦ **H** = h, [**float**]
- ♦ **INAT** = / **1**,
  / **2**,
  / **3**,
  - ♦ **EP** = ep, [**float**]

) # *fin MB_VMAE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en ciaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.8 Classe MB_DPSE

La classe **MB_DPSE** permet d'affecter une loi de comportement de type Drucker-Prager sans écrouissage aux éléments de mécanique bidimensionnelle.

dpse = **MB_DPSE**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]
- ♦ **INAT** = / **1**,
  / **2**,
  / **3**,
  - ♦ **EP** = ep, [**float**]

) # *fin MB_DPSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.

C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.9 Classe MB_DPAE

La classe **MB_DPAE** permet d'affecter une loi de comportement de type Drucker-Prager avec écrouissage aux éléments de mécanique bidimensionnelle.

```
dpae = MB_DPAE(
    ♦ RO = ro, [float]
    ♦ YOUNG = young, [float]
    ♦ POISS = poiss, [float]
    ♦ C = c, [float]
    ♦ PHI = phi, [float]
    ♦ PSI = psi, [float]
    ♦ XHI = xhi, [float]
    ♦ INAT = / 1,
             / 2,
             / 3,
               ♦ EP = ep, [float]
    ) # fin MB_DPAE
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
XHI : réel donnant la valeur du paramètre d'écrouissage.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.10 Classe MB_CP

La classe **MB_CP** permet d'affecter une loi de comportement de type critère parabolique aux éléments de mécanique bidimensionnelle.

```
cp = MB_CP(
    ♦ RO = ro, [float]
    ♦ YOUNG = young, [float]
```

♦ **POISS** = poiss, [**float**]
♦ **RC** = rc, [**float**]
♦ **RT** = rt, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin MB_CP*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.11 Classe MB_VE

La classe **MB_VE** permet d'affecter une loi de comportement de type Vermeer aux éléments de mécanique bidimensionnelle.

ve = **MB_VE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **EPSO** = epso, [**float**]
♦ **PHICV** = phicv, [**float**]
♦ **PHIP** = phip, [**float**]
♦ **BETA** = beta, [**float**]
♦ **EPSCO** = epsco, [**float**]
♦ **PO** = po, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin MB_VE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
EPSO : réel donnant la valeur de la déformation volumique élastique initiale.
PHICV : réel donnant la valeur de l'angle de frottement à l'état critique.
PHIP : réel donnant la valeur de l'angle de frottement au pic.
BETA : réel donnant la valeur du paramètre du modèle.

EPSCO : réel donnant la valeur du paramètre du modèle.
PO : réel donnant la valeur de la pression de référence.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.12 Classe MB_NO

La classe **MB_NO** permet d'affecter une loi de comportement de type Nova aux éléments de mécanique bidimensionnelle.

no = **MB_NO**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **BO** = bo, [**float**]
♦ **LO** = lo, [**float**]
♦ **M** = m, [**float**]
♦ **L** = l, [**float**]
♦ **D** = d, [**float**]
♦ **MM** = mm, [**float**]
♦ **MU** = mu, [**float**]
♦ **PCO** = pco, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin MB_NO*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
BO : réel donnant la valeur de la pente initiale de déchargement.
LO : réel donnant la valeur de la pente initiale de contrainte-déformation.
M : réel donnant la valeur du paramètre du modèle.
L : réel donnant la valeur du paramètre du modèle.
D : réel donnant la valeur du paramètre du modèle.
MM : réel donnant la valeur de la pente de la droite Q/P.
MU : réel donnant la valeur du paramètre du modèle.
PCO : réel donnant la valeur du paramètre définissant la surface de charge initiale.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.13 Classe MB_CCM

La classe **MB_CCM** permet d'affecter une loi de comportement de type Cam Clay modifié aux éléments de mécanique bidimensionnelle.

ccm = **MB_CCM**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **ALOE** = aloe, [**float**]
- ♦ **AKOE** = akoe, [**float**]
- ♦ **AMC** = amc, [**float**]
- ♦ **OED** = oed, [**float**]
- ♦ **PCO** = pco, [**float**]
- ♦ **INAT** = / **1**,
  / **2**,
  / **3**,
  ♦ **EP** = ep, [**float**]

) # *fin MB_CCM*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
ALOE : réel donnant la valeur de la pente de la courbe de consolidation vierge.
AKOE : réel donnant la valeur de la pente des courbes charge-décharge.
AMC : réel donnant la valeur de la pente de la courbe d'état critique.
OED : réel donnant la valeur de l'indice des vides initial.
PCO : réel donnant la valeur de la pression de préconsolidation initiale.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.14 Classe MB_PH

La classe **MB_PH** permet d'affecter une loi de comportement de type Prévost-Hoeg aux éléments de mécanique bidimensionnelle.

ph = **MB_PH**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **AO** = ao, [**float**]
- ♦ **BO** = bo, [**float**]
- ♦ **INAT** = / **1**,
  / **2**,
  / **3**,
  ♦ **EP** = ep, [**float**]

) # *fin MB_PH*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
AO : réel donnant la valeur du paramètre du modèle.
BO : réel donnant la valeur du paramètre du modèle.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.15 Classe MB_CO

La classe **MB_CO** permet d'affecter une loi de comportement de type critère orienté aux éléments de mécanique bidimensionnelle.

co = **MB_CO**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **C** = c, [**float**]
- **PHI** = phi, [**float**]
- **PSI** = psi, [**float**]
- **ALPHA** = alpha, [**float**]
- **INAT** = / **1**,
  / **2**,
  / **3**,
  - **EP** = ep, [**float**]

) # *fin MB_CO*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
ALPHA : réel donnant la valeur de l'angle par rapport à l'axe Ox.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.16 Classe MB_HB

La classe **MB_HB** permet d'affecter une loi de comportement de type Hoek-Brown aux éléments de mécanique bidimensionnelle.

hb = **MB_HB**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **SU** = su, [**float**]
- **S** = s, [**float**]
- **M** = m, [**float**]
- **INAT** = / **1**,
  / **2**,
  / **3**,
  - **EP** = ep, [**float**]

) # *fin MB_HB*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
SU : réel donnant la valeur de la contrainte à la rupture de la roche saine.
S : réel donnant la valeur du coefficient de fracturation.
M : réel donnant la valeur du paramètre de forme.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.17 Classe MB_ME

La classe **MB_ME** permet d'affecter une loi de comportement de type Mélanie aux éléments de mécanique bidimensionnelle.

me = **MB_ME**(
- **RO** = ro, [**float**]
- **E1** = e1, [**float**]
- **E2** = e2, [**float**]
- **P1** = p1, [**float**]
- **P2** = p2, [**float**]
- **G2** = g2, [**float**]
- **TETA** = teta, [**float**]
- **ALPHA** = alpha, [**float**]
- **OED** = oed, [**float**]
- **SIVO** = sivo, [**float**]
- **SIPO** = sipo, [**float**]
- **CPSC** = cpsc, [**float**]
- **CPNC** = cpnc, [**float**]
- **JTA** = jta, [**float**]
- **TOLC** = tolc, [**float**]
- **INAT** = / **1**,
  / **2**,
  / **3**,

) # *fin MB_ME*

RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young dans la direction 1.
E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
ALPHA : réel donnant la valeur de la pente de la courbe de consolidation isotrope.
OED : réel donnant la valeur de l'indice des vides initial.
SIVO : réel donnant la valeur de la contrainte verticale effective initiale.
SIPO : réel donnant la valeur de la pression de préconsolidation.
CPSC : réel donnant la valeur du coefficient de pression des terres au repos du sol surconsolidé.
CPNC : réel donnant la valeur du coefficient de pression des terres au repos du sol normalement consolidé.
JTA : réel donnant la valeur de l'indice de normalité.
TOLC : réel donnant la valeur de la tolérance sur le critère de plasticité.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.

3.6.18 Classe MB_MCSE_EO

La classe **MB_MCSE_EO** permet d'affecter une loi de comportement de type Mohr-Coulomb sans écrouissage avec élasticité orthotrope aux éléments de mécanique bidimensionnelle.

mcse_eo = **MB_MCSE_EO**(
♦ **RO** = ro, [**float**]
♦ **E1** = e1, [**float**]
♦ **E2** = e2, [**float**]
♦ **P1** = p1, [**float**]
♦ **P2** = p2, [**float**]
♦ **G2** = g2, [**float**]
♦ **TETA** = teta, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin MB_MCSE_EO*

RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young dans la direction 1.

E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.

3.6.19 Classe MB_TA

La classe **MB_TA** permet d'affecter une loi de comportement de type Tresca anisotrope aux éléments de mécanique bidimensionnelle.

ta = **MB_TA**(
♦ **RO** = ro, [**float**]
♦ **E1** = e1, [**float**]
♦ **E2** = e2, [**float**]
♦ **P1** = p1, [**float**]
♦ **P2** = p2, [**float**]
♦ **G2** = g2, [**float**]
♦ **TETA** = teta, [**float**]
♦ **A1** = a1, [**float**]
♦ **B1** = b1, [**float**]
♦ **A2** = a2, [**float**]
♦ **B2** = b2, [**float**]
♦ **INAT** = / **1**,
/ **2**,
/ **3**,
) # *fin MB_TA*

RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young dans la direction 1.
E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
A1 : réel donnant la valeur de a_1 dans l'expression $a_1x_2+b_1$ de la variation de la cohésion en compression dans la direction 2.
B1 : réel donnant la valeur de b_1 dans l'expression $a_1x_2+b_1$ de la variation de la cohésion en compression dans la direction 2.
A2 : réel donnant la valeur de a_2 dans l'expression $a_2x_2+b_2$ de la variation de la cohésion en extension dans la direction 2.

B2 : réel donnant la valeur de b_2 dans l'expression $a_2x_2 + b_2$ de la variation de la cohésion en extension dans la direction 2.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.

3.6.20 Classe MB_RBS

La classe **MB_RBS** permet d'affecter une loi de comportement de type matériau renforcé selon le modèle Buhan-Sudret aux éléments de mécanique bidimensionnelle.

```
rbs = MB_RBS(
      ♦ INAT = / 1,
               / 2,
               / 3,
      ♦ RO = ro, [float]
      ♦ YOUNG = young, [float]
      ♦ POISS = poiss, [float]
      ♦ C = c, [float]
      ♦ PHI = phi, [float]
      ♦ PSI = psi, [float]
      ♦ NRENF = / 0,
                / 1,
                / 2,
      ♦ TYP1 = / 0,
               / 1,
               / 2,
               / 3,
      # phase de renforcement 1
      ♦ K1 = k1, [float]
      ♦ SA1 = sa1, [float]
      ♦ S1 = s1, [float]
      ♦ ETA1 = eta1, [float]
      ♦ X1 = x1, [float]
      ♦ Y1 = y1, [float]
      ♦ FV1 = fv1, [float]
      # phase de renforcement 2
      ♦ TYP2 = / 0,
               / 1,
               / 2,
               / 3,
      ♦ K2 = k2, [float]
      ♦ SA2 = sa2, [float]
      ♦ S2 = s2, [float]
      ♦ ETA2 = eta2, [float]
      ♦ X2 = x2, [float]
      ♦ Y2 = y2, [float]
      ♦ FV2 = fv2, [float]
      ) # fin MB_RBS
```

INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
NRENF : entier indiquant le nombre de renforcements à choisir parmi **0**, **1** ou **2**.
TYP1 : entier indiquant le type de la phase de renforcement à choisir parmi :
— **0** : pour un renforcement homogène,
— **1** : pour un renforcement radial,
— **2** : pour un renforcement divergent cylindrique,
— **3** : pour un renforcement divergent sphérique.
K1 : réel donnant la valeur du module d'Young du matériau constituant les inclusions de la phase 1.
SA1 : réel donnant la valeur de la section d'une inclusion de la phase 1.
S1 : réel donnant la valeur de la contrainte limite en traction simple du matériau constituant les inclusions de la phase 1.
ETA1 : réel donnant la valeur du rapport limite en compression / limite en traction des inclusions de la phase 1.
X1 : réel fixé à la valeur 0. sauf si :
— **INAT=1** et **TYP1=0** : angle de la direction des inclusions de la phase 1 avec l'axe horizontal.
— **INAT=1** et **TYP1=1** : coordonnée x du point de convergence des inclusions de la phase 1.
— **INAT=2** et **TYP1=2** : angle de la direction de renforcement de la phase 1 avec l'axe Oz.
— **INAT=2** et **TYP1=3** : coordonnée z du point de convergence des inclusions.
Y1 : réel fixé à la valeur 0. sauf si :
— **INAT=1** et **TYP1=1** : coordonnée y du point de convergence des inclusions de la phase 1.
FV1 : réel donnant la valeur de la fraction volumique des inclusions de la phase 1
TYP2 : idem **TYP1** pour la phase 2.
K2 : idem **K1** pour la phase 2.
SA2 : idem **SA1** pour la phase 2.
S2 : idem **S1** pour la phase 2.
ETA2 : idem **ETA1** pour la phase 2.
X2 : idem **X1** pour la phase 2.
Y2 : idem **Y1** pour la phase 2.
FV2 : idem **FV1** pour la phase 2.

3.6.21 Classe MB_WWS

La classe **MB_WWS** permet d'affecter une loi de comportement de type Willam-Warnke standard aux éléments de mécanique bidimensionnelle.

wws = **MB_WWS**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **FC** = fc, [**float**]
- **FT** = ft, [**float**]
- **FBC** = fbc, [**float**]
- **A0** = a0, [**float**]
- **B0** = b0, [**float**]
- **KAPPA** = kappa, [**float**]
- **INAT** = / **1**,
  / **2**,
  / **3**,
  - **EP** = ep, [**float**]

) # *fin MB_WWS*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.22 Classe MB_WWM

La classe **MB_WWM** permet d'affecter une loi de comportement de type Willam-Warnke modifié aux éléments de mécanique bidimensionnelle.

wwm = **MB_WWM**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **FC** = fc, [**float**]
- **FT** = ft, [**float**]
- **FBC** = fbc, [**float**]
- **SIG** = sig, [**list**<**float**>]
- **A0** = a0, [**float**]
- **B0** = b0, [**float**]
- **KAPPA** = kappa, [**float**]
- **INAT** = / **1**,
  / **2**,

```
                / 3,
                  ♦ EP = ep, [float]
) # fin MB_WWM
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
SIG : liste de 3 réels donnant les valeurs des contraintes triaxiales de ruine.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.23 Classe MB_EDI

La classe **MB_EDI** permet d'affecter une loi de comportement de type élasticité avec dilatance isotrope aux éléments de mécanique bidimensionnelle.

```
edi = MB_EDI(
    ♦ RO = ro, [float]
    ♦ YOUNG = young, [float]
    ♦ POISS = poiss, [float]
    ♦ CKA = cka [float]
    ♦ CKB = ckb, [float]
    ♦ CKC = ckc, [float]
    ♦ INAT = / 1,
             / 2,
             / 3,
               ♦ EP = ep, [float]
) # fin MB_EDI
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
CKA : réel donnant la valeur du coefficient affecté au cisaillement.
CKB : réel donnant la valeur du coefficient affecté à la variation de volume.
CKC : réel donnant la valeur du coefficient affecté au troisième invariant.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant l'épaisseur de la structure.

3.6.24 Classe MB_BAO

La classe **MB_BAO** permet d'affecter un modèle de comportement à composantes (boîte à outils) aux éléments de mécanique bidimensionnelle.

```
bao = MB_BAO(
      ♦ INAT = / 1,
               / 2,
               / 3,
                 ♦ EP = ep, [float]
      ◊ RHO = rho, [float]
      ◊ ELAS = elas, [CMP_ELAS]
      ◊ CRT = crt, [CMP_CRT]
      ◊ POT = pot, [CMP_POT]
      ◊ ECR = ecr, [CMP_ECR]
      ◊ CRT2 = crt2, [CMP_CRT]
      ◊ POT2 = pot2, [CMP_POT]
      ◊ ECR2 = ecr2, [CMP_ECR]
      ◊ RENF = renf, [CMP_RENF]
      ) # fin MB_BAO
```

INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
RHO : réel donnant la valeur de la masse volumique.
EP : réel donnant l'épaisseur de la structure.
ELAS : objet de type **CMP_ELAS** définissant la type d'élasticité.
CRT : objet de type **CMP_CRT** définissant le critère du premier mécanisme plastique.
POT : objet de type **CMP_POT** définissant le potentiel du premier mécanisme plastique.
ECR : objet de type **CMP_ECR** définissant la loi d'écrouissage du premier mécanisme plastique.
CRT2 : objet de type **CMP_CRT** définissant le critère du deuxième mécanisme plastique.
POT2 : objet de type **CMP_POT** définissant le potentiel du deuxième mécanisme plastique.
ECR2 : objet de type **CMP_ECR** définissant la loi d'écrouissage du deuxième mécanisme plastique.
RENF : objet de type **CMP_RENF** définissant la prise en compte par homogénéisation d'inclusions de renforcement.

3.6.25 Classe CMP_ELAS

La classe **CMP_ELAS** permet de définir le type d'élasticité intervenant dans un modèle à composantes.

```
elas = CMP_ELAS(
       ♦ NELAS = / 0,
                 ♦ PARAM = param, [ELAS_LI]
                 / 1,
                 ♦ PARAM = param, [ELAS_LIV]
```

```
                / 4,
                ♦ PARAM = param, [ELAS_FC]
                / 5,
                ♦ PARAM = param, [ELAS_HSM]
                / 24,
                ♦ PARAM = param, [ELAS_ANL]
    ) # fin CMP_ELAS
```

3.6.26 Classe ELAS_LI

La classe **ELAS_LI** permet de définir une loi d'élasticité linéaire isotrope.

```
eli = ELAS_LI(
    ♦ YOUNG = young, [float]
    ♦ POISS = poiss, [float]
    ) # fin ELAS_LI
```

3.6.27 Classe ELAS_LIV

La classe **ELAS_LIV** permet de définir une loi d'élasticité linéaire isotrope variant avec la profondeur.

```
eliv = ELAS_LIV(
    ♦ YOUNGZ = youngz, [float]
    ♦ POISSZ = poissz, [float]
    ♦ DELTAYG = deltayg, [float]
    ♦ DELTAP = deltap, [float]
    ) # fin ELAS_LIV
```

3.6.28 Classe ELAS_FC

La classe **ELAS_FC** permet de définir une loi d'élasticité isotrope non linéaire de type Faher Carter.

```
elifc = ELAS_FC(
    ♦ N = n, [int]
    ♦ NUZERO = nuzero, [float]
    ♦ F = f, [float]
    ♦ G = g, [float]
    ♦ C = c, [float]
    ♦ PREF = pref, [float]
    ♦ COHES = cohes, [float]
    ♦ PHI = phi, [float]
    ) # fin ELAS_FC
```

3.6.29 Classe ELAS_HSM

La classe **ELAS_HSM** permet de définir la loi d'élasticité isotrope non linéaire du Hardening soil model.

elhsm = **ELAS_HSM**(
- ♦ **EUR** = eur, [**float**]
- ♦ **NUUR** = nuur, [**float**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **M** = m, [**int**]
- ♦ **PREF** = pref, [**float**]

) # *fin ELAS_HSM*

3.6.30 Classe ELAS_ANL

La classe **ELAS_ANL** permet de définir une loi d'élasticité isotrope transverse non linéaire (ANL ou Gilleron).

elanl = **ELAS_ANL**(
- ♦ **GREF** = gref, [**float**]
- ♦ **A** = a, [**float**]
- ♦ **R** = r, [**float**]
- ♦ **GMIN_GO** = gmin_go, [**float**]
- ♦ **XSI** = xsi, [**float**]
- ♦ **BETA** = beta, [**float**]
- ♦ **NU0** = nu0, [**float**]
- ♦ **GUR_GO** = gur_go, [**float**]
- ♦ **NU_UR** = nu_ur, [**float**]
- ♦ **N** = n, [**float**]
- ♦ **M_MISO** = m_miso, [**float**]
- ♦ **THETA** = theta, [**float**]
- ◊ **PHI** = phi, [**float**]

) # *fin ELAS_ANL*

3.6.31 Classe CMP_CRT

La classe **CMP_CRT** permet de définir le critère d'un mécanisme plastique intervenant dans un modèle à composantes.

crt = **CMP_CRT**(
- ♦ **ICRIT** = / **1**,
  - ♦ **PARAM** = param, [**FNC_T**]
  - / **2**,
  - ♦ **PARAM** = param, [**FNC_VM**]
  - / **3**,
  - ♦ **PARAM** = param, [**FNC_C**]
  - / **4**,
  - ♦ **PARAM** = param, [**FNC_MC**]

```
                / 15,
                ♦ PARAM = param, [FNC_MCV]
                / 5,
                ♦ PARAM = param, [FNC_DP]
                / 6,
                ♦ PARAM = param, [FNC_CCM]
                / 11,
                ♦ PARAM = param, [FNC_CO]
                / 25,
                ♦ PARAM = param, [FNC_VMC]
                / 27,
                ♦ PARAM = param, [FNC_DPC]
                / 31,
                ♦ PARAM = param, [FNC_HSM]
    ) # fin CMP_CRT
```

3.6.32 Classe FNC_T

La classe **FNC_T** permet de définir une fonction de contraintes de type Tresca.

```
ft = FNC_T(
    ♦ C = c, [float]
    ) # fin FNC_T
```

3.6.33 Classe FNC_VM

La classe **FNC_VM** permet de définir une fonction de contraintes de type Von Mises.

```
fvm = FNC_VM(
      ♦ K = k, [float]
      ) # fin FNC_VM
```

3.6.34 Classe FNC_C

La classe **FNC_C** permet de définir une fonction de contraintes de type Coulomb.

```
fc = FNC_C(
    ♦ PHI = phi, [float]
    ) # fin FNC_C
```

3.6.35 Classe FNC_MC

La classe **FNC_MC** permet de définir une fonction de contraintes de type Mohr-Coulomb.

```
fmc = FNC_MC(
      ♦ C = c, [float]
```

♦ **PHI** = phi, [**float**]
) # *fin FNC_MC*

3.6.36 Classe FNC_MCV

La classe **FNC_MCV** permet de définir une fonction de contraintes de type Mohr-Coulomb variant avec la profondeur.

fmcv = **FNC_MCV**(
♦ **C0** = c0, [**float**]
♦ **DELTAC** = deltac, [**float**]
♦ **PHI0** = phi0, [**float**]
♦ **DELTAPHI** = deltaphi, [**float**]
) # *fin FNC_MCV*

3.6.37 Classe FNC_DP

La classe **FNC_DP** permet de définir une fonction de contraintes de type Drucker Prager.

fdp = **FNC_DP**(
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
) # *fin FNC_DP*

3.6.38 Classe FNC_CCM

La classe **FNC_CCM** permet de définir une fonction de contraintes de type Cam-Clay modifié.

fccm = **FNC_CCM**(
♦ **PHI** = phi, [**float**]
♦ **PC0** = pc0, [**float**]
) # *fin FNC_CCM*

3.6.39 Classe FNC_CO

La classe **FNC_CO** permet de définir une fonction de contraintes de type critère orienté.

fco = **FNC_CO**(
♦ **PHI** = phi, [**float**]
♦ **ALPHA** = alpha, [**float**]
♦ **NX** = nx, [**float**]
♦ **NY** = ny, [**float**]
♦ **NZ** = nz, [**float**]
) # *fin FNC_CO*

3.6.40 Classe FNC_VMC

La classe **FNC_VMC** permet de définir une fonction de contraintes de type von Mises cyclique.

fvmc = **FNC_VMC**(
♦ **K** = k, [**float**]
) # *fin FNC_VMC*

3.6.41 Classe FNC_DPC

La classe **FNC_DPC** permet de définir une fonction de contraintes de type Drucker Prager cyclique.

fdpc = **FNC_DPC**(
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
) # *fin FNC_DPC*

3.6.42 Classe FNC_HSM

La classe **FNC_HSM** permet de définir une fonction de contraintes de type hardening soil model.

fhsm = **FNC_HSM**(
♦ **C** = c, [**float**]
♦ **PHIP** = phip, [**float**]
♦ **PSIP** = psip, [**float**]
♦ **M** = m, [**float**]
♦ **PREF** = pref, [**float**]
♦ **RF** = rf, [**float**]
♦ **EUR** = eur, [**float**]
♦ **EREF50** = eref50, [**float**]
) # *fin FNC_HSM*

3.6.43 Classe CMP_POT

La classe **CMP_POT** permet de définir le potentiel d'un mécanisme plastique intervenant dans un modèle à composantes.

pot = **CMP_POT**(
♦ **IPOT** = / **1**,
♦ **PARAM** = param, [**FNC_T**]
/ **2**,
♦ **PARAM** = param, [**FNC_VM**]
/ **3**,
♦ **PARAM** = param, [**FNC_C**]
/ **4**,
♦ **PARAM** = param, [**FNC_MC**]
/ **15**,

```
                    ♦ PARAM = param, [FNC_MCV]
                    / 5,
                    ♦ PARAM = param, [FNC_DP]
                    / 6,
                    ♦ PARAM = param, [FNC_CCM]
                    / 11,
                    ♦ PARAM = param, [FNC_CO]
                    / 25,
                    ♦ PARAM = param, [FNC_VMC]
                    / 27,
                    ♦ PARAM = param, [FNC_DPC]
                    / 31,
                    ♦ PARAM = param, [FNC_HSM]
    ) # fin CMP_POT
```

3.6.44 Classe CMP_ECR

La classe **CMP_ECR** permet de définir la loi d'écrouissage d'un mécanisme plastique intervenant dans un modèle à composantes.

```
ecr = CMP_ECR(
      ♦ IECR = / # pour ICRIT = 2
                 / 1,
                 ♦ PARAM = param, [ECR_L]
                 / 2,
                 ♦ PARAM = param, [ECR_PH]
                 / 3,
                 ♦ PARAM = param, [ECR_PHM]
               / # pour ICRIT = 5
                 / 1,
                 ♦ PARAM = param, [ECR_NMC]
                 / 3,
                 ♦ PARAM = param, [ECR_VM]
                 / 4,
                 ♦ PARAM = param, [ECR_VMV]
               / # pour ICRIT = 6
                 / 1,
                 ♦ PARAM = param, [ECR_V]
                 / 3,
                 ♦ PARAM = param, [ECR_VD]
                 / 6,
                 ♦ PARAM = param, [ECR_VPP]
                 / 7,
                 ♦ PARAM = param, [ECR_VRS]
               / # pour ICRIT = 25
                 / 1,
                 ♦ PARAM = param, [ECR_CH]
               / # pour ICRIT = 27
                 / 1,
```

```
                ♦ PARAM = param, [ECR_CH]
              / # pour ICRIT = 31
                / 1,
                ♦ PARAM = param, [ECR_HSM]
    ) # fin CMP_ECR
```

3.6.45 Classe ECR_L

La classe **ECR_L** permet de définir une loi d'écrouissage linéaire.

```
el = ECR_L(
    ♦ A = a, [float]
    ) # fin ECR_L
```

3.6.46 Classe ECR_PH

La classe **ECR_PH** permet de définir une loi d'écrouissage de Prevost et Hoeg.

```
eph = ECR_PH(
      ♦ A = a, [float]
      ♦ B = b, [float]
      ) # fin ECR_PH
```

3.6.47 Classe ECR_PHM

La classe **ECR_PHM** permet de définir une loi d'écrouissage de Prevost et Hoeg modifiée.

```
ephm = ECR_PHM(
       ♦ A = a, [float]
       ♦ B = b, [float]
       ) # fin ECR_PHM
```

3.6.48 Classe ECR_NMC

La classe **ECR_NMC** permet de définir une loi d'écrouissage négatif à module constant.

```
enmc = ECR_NMC(
       ♦ H = h, [float]
       ) # fin ECR_NMC
```

3.6.49 Classe ECR_VM

La classe **ECR_VM** permet de définir une loi d'écrouissage volumique variable.

evm = **ECR_VM**(
♦ **HMAX** = hmax, [**float**]
♦ **HRES** = hres, [**float**]
♦ **XMAX** = xmax, [**float**]
) # *fin ECR_VM*

3.6.50 Classe ECR_VMV

La classe **ECR_VMV** permet de définir une loi d'écrouissage volumique variable (variante).

evmv = **ECR_VMV**(
♦ **HMAX** = hmax, [**float**]
♦ **HFIN** = hfin, [**float**]
♦ **EPSMAX** = epsmax, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin ECR_VMV*

3.6.51 Classe ECR_V

La classe **ECR_V** permet de définir une loi d'écrouissage volumique.

ev = **ECR_V**(
♦ **LAMBDA** = lambda, [**float**]
♦ **KAPPA** = kappa, [**float**]
♦ **E0** = e0, [**float**]
) # *fin ECR_V*

3.6.52 Classe ECR_VD

La classe **ECR_VD** permet de définir une loi d'écrouissage volumique et déviatorique.

evd = **ECR_VD**(
♦ **LAMBDA** = lambda, [**float**]
♦ **KAPPA** = kappa, [**float**]
♦ **E0** = e0, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin ECR_VD*

3.6.53 Classe ECR_VPP

La classe **ECR_VPP** permet de définir une loi d'écrouissage volumique avec pression de préconsolidation initiale linéaire avec la profondeur.

evpp = **ECR_VPP**(
♦ **LAMBDA** = lambda, [**float**]
♦ **KAPPA** = kappa, [**float**]

♦ **E0** = e0, [**float**]
♦ **PCZ** = pcz, [**float**]
♦ **DPC** = dpc, [**float**]
) # *fin ECR_VPP*

3.6.54 Classe ECR_VRS

La classe **ECR_VRS** permet de définir une loi d'écrouissage volumique avec prise en compte d'un rapport de surconsolidation OCR.

evrs = **ECR_VRS**(
♦ **LAMBDA** = lambda, [**float**]
♦ **KAPPA** = kappa, [**float**]
♦ **E0** = e0, [**float**]
♦ **OCR** = ocr, [**float**]
) # *fin ECR_VRS*

3.6.55 Classe ECR_CH

La classe **ECR_CH** permet de définir une loi d'écrouissage de Chaboche.

ech = **ECR_CH**(
♦ **C** = c, [**float**]
♦ **D** = d, [**float**]
) # *fin ECR_CH*

3.6.56 Classe ECR_HSM

La classe **ECR_HSM** permet de définir une loi d'écrouissage pour le hardening soil model.

ehsm = **ECR_HSM**(
) # *fin ECR_HSM*

3.6.57 Classe CMP_RENF

La classe **CMP_RENF** permet de définir la prise en compte par homogénéisation d'inclusions de renforcement intervenant dans un modèle à composantes.

renf = **CMP_RENF**(
♦ **ICOMP** = / **1**,
♦ **PARAMC** = paramc, [**RENF_EL**]
/ **2**,
♦ **PARAMC** = paramc, [**RENF_PP**]
♦ **IGEOM** = / **1**,
♦ **PARAMG** = paramc, [**RENF_H**]
/ **2**,

```
                    ♦ PARAMG = paramc, [RENF_R]
                    / 3,
                    ♦ PARAMG = paramc, [RENF_DC]
                    / 4,
                    ♦ PARAMG = paramc, [RENF_DS]
    ) # fin CMP_RENF
```

3.6.58 Classe RENF_EL

La classe **RENF_EL** permet de définir un comportement des inclusions de type élastique linéaire.

```
rel = RENF_EL(
      ♦ X = x, [float]
      ♦ SA = sa, [float]
      ) # fin RENF_EL
```

3.6.59 Classe RENF_PP

La classe **RENF_PP** permet de définir un comportement des inclusions de type élastique linéaire et plasticité parfaite.

```
rpp = RENF_PP(
      ♦ X = x, [float]
      ♦ SA = sa, [float]
      ♦ S = s, [float]
      ♦ ETA = eta, [float]
      ) # fin RENF_PP
```

3.6.60 Classe RENF_H

La classe **RENF_H** permet de définir une disposition spatiale des inclusions de type renforcement homogène.

```
rh = RENF_H(
     ♦ FV = fv, [float]
     ◊ ALPHA = alpha, [float]
     ◊ BETA = beta, [float]
     ) # fin RENF_H
```

3.6.61 Classe RENF_R

La classe **RENF_R** permet de définir une disposition spatiale des inclusions de type renforcement radial.

```
rr = RENF_R(
     ♦ FV = fv, [float]
```

◊ **X** = x, [**float**]
◊ **Y** = y, [**float**]
◊ **Z** = z, [**float**]
◊ **ALPHA** = alpha, [**float**]
◊ **BETA** = beta, [**float**]
) # *fin RENF_R*

3.6.62 Classe RENF_DC

La classe **RENF_DC** permet de définir une disposition spatiale des inclusions de type renforcement divergent cylindrique.

rdc = **RENF_DC**(
♦ **FV** = fv, [**float**]
♦ **ALPHA** = alpha, [**float**]
◊ **X** = x, [**float**]
◊ **Y** = y, [**float**]
◊ **Z** = z, [**float**]
◊ **BETA** = beta, [**float**]
◊ **GAMMA** = gamma, [**float**]
) # *fin RENF_DC*

3.6.63 Classe RENF_DS

La classe **RENF_DS** permet de définir une disposition spatiale des inclusions de type renforcement divergent sphérique.

rds = **RENF_DS**(
♦ **FV** = fv, [**float**]
◊ **X** = x, [**float**]
◊ **Y** = y, [**float**]
♦ **Z** = z, [**float**]
) # *fin RENF_DS*

3.7 Données relatives au modèle de mécanique tridimensionnelle

3.7.1 Classe MT

La classe **MT** permet de définir les caractéristiques d'un groupe d'éléments de mécanique tridimensionnelle.

mt = **MT**(
♦ **NOMG** = nomg, [**str**]
♦ **ACTI** = / **'A'**,
♦ **IMOD** = / **1**,

```
                    ♦ CARA = cara, [MT_ELI]
                  / 2,
                    ♦ CARA = cara, [MT_ELO]
                  / 5,
                    ♦ CARA = cara, [MT_BJA]
                  / 10,
                    ♦ CARA = cara, [MT_MCSE]
                  / 11,
                    ♦ CARA = cara, [MT_VMSE]
                  / 12,
                    ♦ CARA = cara, [MT_VMAE]
                  / 13,
                    ♦ CARA = cara, [MT_DPSE]
                  / 14,
                    ♦ CARA = cara, [MT_DPAE]
                  / 15,
                    ♦ CARA = cara, [MT_CP]
                  / 16,
                    ♦ CARA = cara, [MT_VE]
                  / 17,
                    ♦ CARA = cara, [MT_NO]
                  / 18,
                    ♦ CARA = cara, [MT_CCM]
                  / 19,
                    ♦ CARA = cara, [MT_PH]
                  / 20,
                    ♦ CARA = cara, [MT_CO]
                  / 24,
                    ♦ CARA = cara, [MT_HB]
                  / 43,
                    ♦ CARA = cara, [MT_RBS]
                  / 47,
                    ♦ CARA = cara, [MT_WWS]
                  / 48,
                    ♦ CARA = cara, [MT_WWM]
                  / 88,
                    ♦ CARA = cara, [MT_EDI]
                  / 10000,
                    ♦ CARA = cara, [MT_BAO]
          / 'I',
) # fin MT
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **5**, **10**, **11**, **12**, **13**, **14**, **15**, **16**, **17**, **18**, **19**, **20**, **24**, **43**, **47**, **48**, **88**.
CARA : objet de type à choisir parmi :

- **MT_ELI** pour une loi de type élasticite lineaire isotrope,
- **MT_ELO** pour une loi de type élasticite lineaire orthotrope,

· **MT_BJA** pour une loi de type béton au jeune âge,
· **MT_MCSE** pour une loi de type Mohr-Coulomb sans écrouissage,
· **MT_VMSE** pour une loi de type Von Mises sans écrouissage,
· **MT_VMAE** pour une loi de type Von Mises avec écrouissage,
· **MT_DPSE** pour une loi de type Drucker-Prager sans écrouissage,
· **MT_DPAE** pour une loi de type Drucker-Prager avec écrouissage,
· **MT_CP** pour une loi de type critère parabolique,
· **MT_VE** pour une loi de type Vermeer,
· **MT_NO** pour une loi de type Nova,
· **MT_CCM** pour une loi de type Cam Clay modifié,
· **MT_PH** pour une loi de type Prevost-Hoeg,
· **MT_CO** pour une loi de type critère orienté,
· **MT_HB** pour une loi de type Hoek Brown,
· **MT_RBS** pour une loi de type matériau renforcé,
· **MT_WWS** pour une loi de type William-Warnke standard,
· **MT_WWM** pour une loi de type William-Warnke modifié,
· **MT_EDI** pour une loi de type élasticite avec dilatance isotrope,
· **MT_BAO** pour un modèle à composantes,
en cohérence avec le choix effectué pour **IMOD**.

3.7.2 Classe MT_ELI

La classe **MT_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de mécanique tridimensionnelle.

eli = **MT_ELI**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
) # *fin MT_ELI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.

3.7.3 Classe MT_ELO

La classe **MT_ELO** permet d'affecter une loi de comportement de type élasticité linéaire orthotrope aux éléments de mécanique tridimensionnelle.

elo = **MT_ELO**(
♦ **RO** = ro, [**float**]
♦ **E1** = e1, [**float**]
♦ **E2** = e2, [**float**]
♦ **P1** = p1, [**float**]
♦ **P2** = p2, [**float**]
♦ **G2** = g2, [**float**]
♦ **TETA** = teta, [**float**]
♦ **PHI** = phi, [**float**]

) # *fin MT_ELO*

RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young dans la direction 1.
E2 : réel donnant la valeur du module d'Young dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
PHI : réel donnant la valeur de l'angle entre la direction 1 et l'axe d'orthotropie.

3.7.4 Classe MT_BJA

La classe **MT_BJA** permet d'affecter une loi de comportement de type béton au jeune âge aux éléments de mécanique tridimensionnelle.

bja = **MT_BJA**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **DILAT** = dilat, [**float**]
- **RETRA** = retra, [**float**]
- **SEUIL** = seuil, [**float**]
- **HYD** = hyd, [**float**]

) # *fin MT_BJA*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
DILAT : réel donnant la valeur du coefficient de dilatation thermique.
RETRA : réel donnant la valeur du retrait endogène final.
SEUIL : réel donnant la valeur du seuil du matériau durci.
HYD : réel donnant la valeur du degré d'hydratation.

3.7.5 Classe MT_MCSE

La classe **MT_MCSE** permet d'affecter une loi de comportement de type Mohr-Coulomb sans écrouissage aux éléments de mécanique tridimensionnelle.

mcse = **MT_MCSE**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **C** = c, [**float**]
- **PHI** = phi, [**float**]
- **PSI** = psi, [**float**]

) # *fin MT_MCSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.

3.7.6 Classe MT_VMSE

La classe **MT_VMSE** permet d'affecter une loi de comportement de type Von Mises sans écrouissage aux éléments de mécanique tridimensionnelle.

vmse = **MT_VMSE**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **K** = k, [**float**]

) # *fin MT_VMSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en ciaillement simple.

3.7.7 Classe MT_VMAE

La classe **MT_VMAE** permet d'affecter une loi de comportement de type Von Mises avec écrouissage aux éléments de mécanique tridimensionnelle.

vmae = **MT_VMAE**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **K** = k, [**float**]
- **H** = h, [**float**]

) # *fin MT_VMAE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en ciaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.

3.7.8 Classe MT_DPSE

La classe **MT_DPSE** permet d'affecter une loi de comportement de type Drucker-Prager sans écrouissage aux éléments de mécanique tridimensionnelle.

dpse = **MT_DPSE**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]

) # *fin MT_DPSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.

3.7.9 Classe MT_DPAE

La classe **MT_DPAE** permet d'affecter une loi de comportement de type Drucker-Prager avec écrouissage aux éléments de mécanique tridimensionnelle.

dpae = **MT_DPAE**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]
- ♦ **XHI** = xhi, [**float**]

) # *fin MT_DPAE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
XHI : réel donnant la valeur du paramètre d'écrouissage.

3.7.10 Classe MT_CP

La classe **MT_CP** permet d'affecter une loi de comportement de type critère parabolique aux éléments de mécanique tridimensionnelle.

cp = **MT_CP**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]

♦ **RC** = rc, [**float**]
♦ **RT** = rt, [**float**]
) # *fin MT_CP*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple.

3.7.11 Classe MT_VE

La classe **MT_VE** permet d'affecter une loi de comportement de type Vermeer aux éléments de mécanique tridimensionnelle.

ve = **MT_VE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **EPSO** = epso, [**float**]
♦ **PHICV** = phicv, [**float**]
♦ **PHIP** = phip, [**float**]
♦ **BETA** = beta, [**float**]
♦ **EPSCO** = epsco, [**float**]
♦ **PO** = po, [**float**]
) # *fin MT_VE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
EPSO : réel donnant la valeur de la déformation volumique élastique initiale.
PHICV : réel donnant la valeur de l'angle de frottement à l'état critique.
PHIP : réel donnant la valeur de l'angle de frottement au pic.
BETA : réel donnant la valeur du paramètre du modèle.
EPSCO : réel donnant la valeur du paramètre du modèle.
PO : réel donnant la valeur de la pression de référence.

3.7.12 Classe MT_NO

La classe **MT_NO** permet d'affecter une loi de comportement de type Nova aux éléments de mécanique tridimensionnelle.

no = **MT_NO**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **BO** = bo, [**float**]
♦ **LO** = lo, [**float**]

♦ **M** = m, [**float**]
♦ **L** = l, [**float**]
♦ **D** = d, [**float**]
♦ **MM** = mm, [**float**]
♦ **MU** = mu, [**float**]
♦ **PCO** = pco, [**float**]
) # *fin MT_NO*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
BO : réel donnant la valeur de la pente initiale de déchargement.
LO : réel donnant la valeur de la pente initiale de contrainte-déformation.
M : réel donnant la valeur du paramètre du modèle.
L : réel donnant la valeur du paramètre du modèle.
D : réel donnant la valeur du paramètre du modèle.
MM : réel donnant la valeur de la pente de la droite Q/P.
MU : réel donnant la valeur du paramètre du modèle.
PCO : réel donnant la valeur du paramètre définissant la surface de charge initiale.

3.7.13 Classe MT_CCM

La classe **MT_CCM** permet d'affecter une loi de comportement de type Cam Clay modifié aux éléments de mécanique tridimensionnelle.

ccm = **MT_CCM**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **ALOE** = aloe, [**float**]
♦ **AKOE** = akoe, [**float**]
♦ **AMC** = amc, [**float**]
♦ **OED** = oed, [**float**]
♦ **PCO** = pco, [**float**]
) # *fin MT_CCM*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
ALOE : réel donnant la valeur de la pente de la courbe de consolidation vierge.
AKOE : réel donnant la valeur de la pente des courbes charge-décharge.
AMC : réel donnant la valeur de la pente de la courbe d'état critique.
OED : réel donnant la valeur de l'indice des vides initial.
PCO : réel donnant la valeur de la pression de préconsolidation initiale.

3.7.14 Classe MT_PH

La classe **MT_PH** permet d'affecter une loi de comportement de type Prévost-Hoeg aux éléments de mécanique tridimensionnelle.

ph = **MT_PH**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **AO** = ao, [**float**]
- ♦ **BO** = bo, [**float**]

) # *fin MT_PH*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
AO : réel donnant la valeur du paramètre du modèle.
BO : réel donnant la valeur du paramètre du modèle.

3.7.15 Classe MT_CO

La classe **MT_CO** permet d'affecter une loi de comportement de type critère orienté aux éléments de mécanique tridimensionnelle.

co = **MT_CO**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]
- ♦ **U** = u, [**list**<**float**>]

) # *fin MT_CO*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
U : liste de 3 réels donnant les coordonnées du vecteur normal au plan de discontinuité.

3.7.16 Classe MT_HB

La classe **MT_HB** permet d'affecter une loi de comportement de type Hoek-Brown aux éléments de mécanique tridimensionnelle.

hb = **MT_HB**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **SU** = su, [**float**]
- ♦ **S** = s, [**float**]

♦ **M** = m, [**float**]
) # *fin MT_HB*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
SU : réel donnant la valeur de la contrainte à la rupture de la roche saine.
S : réel donnant la valeur du coefficient de fracturation.
M : réel donnant la valeur du paramètre de forme.

3.7.17 Classe MT_RBS

La classe **MT_RBS** permet d'affecter une loi de comportement de type matériau renforcé selon le modèle Buhan-Sudret aux éléments de mécanique tridimensionnelle.

rbs = **MT_RBS**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **NRENF** = / **0**,
/ **1**,
/ **2**,
phase de renforcement 1
♦ **TYP1** = / **0**,
/ **1**,
/ **2**,
/ **3**,
♦ **K1** = k1, [**float**]
♦ **SA1** = sa1, [**float**]
♦ **S1** = s1, [**float**]
♦ **ETA1** = eta1, [**float**]
♦ **X1** = x1, [**float**]
♦ **Y1** = y1, [**float**]
♦ **Z1** = z1, [**float**]
♦ **A1** = a1, [**float**]
♦ **B1** = b1, [**float**]
♦ **C1** = c1, [**float**]
♦ **FV1** = fv1, [**float**]
phase de renforcement 2
♦ **TYP2** = / **0**,
/ **1**,
/ **2**,
/ **3**,
♦ **K2** = k2, [**float**]
♦ **SA2** = sa2, [**float**]
♦ **S2** = s2, [**float**]

♦ **ETA2** = eta2, [**float**]
♦ **X2** = x2, [**float**]
♦ **Y2** = y2, [**float**]
♦ **Z2** = z2, [**float**]
♦ **A2** = a2, [**float**]
♦ **B2** = b2, [**float**]
♦ **C2** = c2, [**float**]
♦ **FV2** = fv2, [**float**]
) # *fin MT_RBS*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
NRENF : entier indiquant le nombre de renforcements à choisir parmi **0**, **1** ou **2**.
TYP1 : entier indiquant le type de la phase de renforcement à choisir parmi :
— **0** : pour un renforcement homogène,
— **1** : pour un renforcement radial,
— **2** : pour un renforcement divergent cylindrique,
— **3** : pour un renforcement divergent sphérique.
K1 : réel donnant la valeur du module d'Young du matériau constituant les inclusions de la phase 1.
SA1 : réel donnant la valeur de la section d'une inclusion de la phase 1.
S1 : réel donnant la valeur de la contrainte limite en traction simple du matériau constituant les inclusions de la phase 1.
ETA1 : réel donnant la valeur du rapport limite en compression / limite en traction des inclusions de la phase 1.
X1 : réel fixé à la valeur 0. sauf si :
— **TYP1=1** : coordonnée x d'un point de l'axe de symétrie de révolution.
— **TYP1=2** : coordonnée x d'un point de l'axe de symétrie de révolution.
— **TYP1=3** : coordonnée x du point de convergence des inclusions.
Y1 : idem **X1** pour la coordonnée y.
Z1 : idem **X1** pour la coordonnée z.
A1 : réel fixé à la valeur 0. sauf si :
— **TYP1=0** : angle entre la projection de la direction de renforcement sur $z = 0$ et l'axe Ox.
— **TYP1=1** : angle entre la projection de l'axe de symétrie sur $z = 0$ et l'axe Ox.
— **TYP1=2** : angle entre la projection de l'axe de symétrie sur $z = 0$ et l'axe Ox.
B1 : réel fixé à la valeur 0. sauf si :
— **TYP1=0** : angle entre la direction de renforcement l'axe horizontal.
— **TYP1=1** : angle entre la direction de l'axe de symétrie et l'horizontale.
— **TYP1=2** : angle entre la direction de l'axe de symétrie et l'horizontale.
C1 : réel fixé à la valeur 0. sauf si :
— **TYP1=2** : angle entre la direction de renforcement et l'axe de symétrie dans un plan méridien.
FV1 : réel donnant la valeur de la fraction volumique des inclusions.
TYP2 : idem **TYP1** pour la phase 2.

K2 : idem **K1** pour la phase 2.
SA2 : idem **SA1** pour la phase 2.
S2 : idem **S1** pour la phase 2.
ETA2 : idem **ETA1** pour la phase 2.
X2 : idem **X1** pour la phase 2.
Y2 : idem **Y1** pour la phase 2.
Z2 : idem **Z1** pour la phase 2.
A2 : idem **A1** pour la phase 2.
B2 : idem **B1** pour la phase 2.
C2 : idem **C1** pour la phase 2.
FV2 : idem **FV1** pour la phase 2.

3.7.18 Classe MT_WWS

La classe **MT_WWS** permet d'affecter une loi de comportement de type Willam-Warnke standard aux éléments de mécanique tridimensionnelle.

wws = **MT_WWS**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **FC** = fc, [**float**]
- **FT** = ft, [**float**]
- **FBC** = fbc, [**float**]
- **A0** = a0, [**float**]
- **B0** = b0, [**float**]
- **KAPPA** = kappa, [**float**]

) # *fin MT_WWS*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.

3.7.19 Classe MT_WWM

La classe **MT_WWM** permet d'affecter une loi de comportement de type Willam-Warnke modifié aux éléments de mécanique tridimensionnelle.

wwm = **MT_WWM**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **FC** = fc, [**float**]

♦ **FT** = ft, [**float**]
♦ **FBC** = fbc, [**float**]
♦ **SIG** = sig, [**list**<**float**>]
♦ **A0** = a0, [**float**]
♦ **B0** = b0, [**float**]
♦ **KAPPA** = kappa, [**float**]
) # *fin MT_WWM*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
SIG : liste de 3 réels donnant les valeurs des contraintes triaxiales de ruine.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.

3.7.20 Classe MT_EDI

La classe **MT_EDI** permet d'affecter une loi de comportement de type élasticité avec dilatance isotrope aux éléments de mécanique tridimensionnelle.

edi = **MT_EDI**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **CKA** = cka [**float**]
♦ **CKB** = ckb, [**float**]
♦ **CKC** = ckc, [**float**]
) # *fin MT_EDI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
CKA : réel donnant la valeur du coefficient affecté au cisaillement.
CKB : réel donnant la valeur du coefficient affecté à la variation de volume.
CKC : réel donnant la valeur du coefficient affecté au troisième invariant.

3.7.21 Classe MT_BAO

La classe **MT_BAO** permet d'affecter un modèle de comportement à composantes (boîte à outils) aux éléments de mécanique tridimensionnelle.

bao = **MT_BAO**(
◊ **RHO** = rho, [**float**]
◊ **ELAS** = elas, [**CMP_ELAS**]

◊ **CRT** = crt, [**CMP_CRT**]
◊ **POT** = pot, [**CMP_POT**]
◊ **ECR** = ecr, [**CMP_ECR**]
◊ **CRT2** = crt2, [**CMP_CRT**]
◊ **POT2** = pot2, [**CMP_POT**]
◊ **ECR2** = ecr2, [**CMP_ECR**]
◊ **RENF** = renf, [**CMP_RENF**]
) # *fin MT_BAO*

RHO : réel donnant la valeur de la masse volumique.
ELAS : objet de type **CMP_ELAS** définissant la type d'élasticité.
CRT : objet de type **CMP_CRT** définissant le critère du premier mécanisme plastique.
POT : objet de type **CMP_POT** définissant le potentiel du premier mécanisme plastique.
ECR : objet de type **CMP_ECR** définissant la loi d'écrouissage du premier mécanisme plastique.
CRT2 : objet de type **CMP_CRT** définissant le critère du deuxième mécanisme plastique.
POT2 : objet de type **CMP_POT** définissant le potentiel du deuxième mécanisme plastique.
ECR2 : objet de type **CMP_ECR** définissant la loi d'écrouissage du deuxième mécanisme plastique.
RENF : objet de type **CMP_RENF** définissant la prise en compte par homogénéisation d'inclusions de renforcement.

3.8 Données relatives au modèle de poutre bidimensionnelle

3.8.1 Classe PB

La classe **PB** permet de définir les caractéristiques d'un groupe d'éléments de poutre bidimensionnelle.

pb = **PB**(
♦ **NOMG** = nomg, [**str**]
♦ **ACTI** = / **'A'**,
♦ **IMOD** = / **1**,
♦ **CARA** = cara, [**PB_ELI**]
/ **'I'**,
) # *fin PB*

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**.
CARA : objet de type à choisir parmi :
· **PB_ELI** pour une loi de type élasticite linéaire isotrope,
en cohérence avec le choix effectué pour **IMOD**.

3.8.2 Classe PB_ELI

La classe **PB_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de poutre bidimensionnelle.

```
eli = PB_ELI(
      ♦ RO = ro, [float]
      ♦ YOUNG = young, [float]
      ♦ POISS = poiss, [float]
      ♦ S = s, [float]
      ♦ SR = sr, [float]
      ♦ VIN = vin, [float]
      ♦ YG = yg, [float]
      ) # fin PB_ELI
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
S : réel donnant la valeur de l'aire de la section.
SR : réel donnant la valeur de la section réduite au cisaillement.
VIN : réel donnant la valeur du moment d'inertie principal de la section.
YG : réel donnant la valeur de l'ordonnée de l'axe des centres de gravité de la section.

3.9 Données relatives au modèle de poutre tridimensionnelle

3.9.1 Classe PT

La classe **PT** permet de définir les caractéristiques d'un groupe d'éléments de poutre tridimensionnelle.

```
pt = PT(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                 ♦ IMOD = / 1,
                            ♦ CARA = cara, [PT_STD_ELI]
                          / 5,
                            ♦ CARA = cara, [PT_MF]
               / 'I',
      ) # fin PT
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **5**.
CARA : objet de type à choisir parmi :
- **PT_STD_ELI** pour un modèle de poutre standard en élasticité linéaire isotrope,
- **PT_MF** pour un modèle de poutre multifibre,

en cohérence avec le choix effectué pour **IMOD**.

3.9.2 Classe PT_STD_ELI

La classe **PT_STD_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de poutre tridimensionnelle.

eli = **PT_STD_ELI**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **S** = s, [**float**]
- **S2** = s2, [**float**]
- **S3** = s3, [**float**]
- **VI1** = vi1, [**float**]
- **VI2** = vi2, [**float**]
- **VI3** = vi3, [**float**]
- **YG** = yg, [**float**]
- **ZG** = zg, [**float**]
- **YC** = yc, [**float**]
- **ZC** = zc, [**float**]
- **V** = v, [**list<float**]

) # *fin PT_STD_ELI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
S : réel donnant la valeur de l'aire de la section droite.
S2 : réel donnant la valeur de la section réduite au cisaillement dans la direction 2.
S3 : réel donnant la valeur de la section réduite au cisaillement dans la direction 3.
VI1 : réel donnant la valeur du moment d'inertie de torsion.
VI2 : réel donnant la valeur du moment d'inertie principal par rapport à l'axe 2.
VI3 : réel donnant la valeur du moment d'inertie principal par rapport à l'axe 3.
YG : réel donnant la valeur de la coordonnée suivant l'axe 2 de l'axe de centres de gravité des section.
ZG : réel donnant la valeur de la coordonnée suivant l'axe 3 de l'axe de centres de gravité des section.
YC : réel donnant la valeur de la coordonnée suivant l'axe 2 de l'axe de centres de torsion des section.
ZC : réel donnant la valeur de la coordonnée suivant l'axe 3 de l'axe de centres de torsion des section.
V : liste de 3 réels donnant les coordonnées du vecteur directeur de l'axe 2.

3.9.3 Classe PT_MF

La classe **PT_MF** permet de définir la géométrie et les matériaux constitutifs d'une poutre multi-fibres.

mf = **PT_MF**(
- **NFIBR** = nfibr, [**int**]
- **NLOI** = nloi, [**int**]
- **LOIS** = lois, [**list<PT_LF>**]
- **YC** = yc, [**float**]
- **ZC** = zc, [**float**]
- **VITORS** = vitors, [**float**]
- **FBRS** = fbrs, [**list<PT_CF>**]

♦ **V** = v, [**list**<**float**>]
) # *fin PT_MF*

NFIBR : entier donnant le nombre de fibres.
NLOI : entier donnant le nombre de lois de comportement différentes.
LOIS : liste d'objets de type **PT_LF** donnant les **NLOI** lois de comportements.
YC : réel donnant la coordonnée suivant l'axe 2 de l'axe des centres de torsion des sections.
ZC : réel donnant la coordonnée suivant l'axe 3 de l'axe des centres de torsion des sections.
VITORS : réel donnant le moment d'inertie de torsion (0. s'il doit être calculé).
FBRS : liste d'objets de type **PT_CF** donnant les caractéristiques des fibres.
V : liste de 3 réels donnant les coordonnées du vecteur directeur de l'axe 2.

3.9.4 Classe PT_LF

La classe **PT_LF** permet d'affecter une loi de comportement sur une fibre d'élément de poutre multi-fibres.

```
lf = PT_LF(
    ♦ IMODF = / 1,
                 ♦ CARA = cara, [PT_LF_ELI]
              / 11,
                 ♦ CARA = cara, [PT_LF_VMSE]
              / 12,
                 ♦ CARA = cara, [PT_LF_VMAE]
              / 15,
                 ♦ CARA = cara, [PT_LF_CP]
              / 47,
                 ♦ CARA = cara, [PT_LF_WWS]
              / 48,
                 ♦ CARA = cara, [PT_LF_WWM]
    ) # fin PT_LF
```

IMODF : entier donnant le type de modèle mécanique utilisé pour la fibre à choisir parmi **1**, **11**, **12**, **15**, **47**, **48**.
CARA : objet de type à choisir parmi :
- **PT_LF_ELI** pour une loi de type élasticité linéaire isotrope,
- **PT_LF_VMSE** pour une loi de type Von Mises sans écrouissage,
- **PT_LF_VMAE** pour une loi de type Von Mises avec écrouissage,
- **PT_LF_CP** pour une loi de type critère parabolique,
- **PT_LF_WWS** pour une loi de type Willam-Warnke standard,
- **PT_LF_WWM** pour une loi de type Willam-Warnke modifié

en cohérence avec le choix effectué pour **IMODF**.

3.9.5 Classe PT_LF_ELI

La classe **PT_LF_ELI** permet de définir une loi de comportement de type élasticité linéaire isotrope pour une fibre d'élément de poutre multi-fibre.

eli = **PT_LF_ELI**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
) # *fin PT_LF_ELI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.

3.9.6 Classe PT_LF_VMSE

La classe **PT_LF_VMSE** permet de définir une loi de comportement de type Von Mises Sans Ecrouissage pour une fibre d'élément de poutre multi-fibre.

vmse = **PT_LF_VMSE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **K** = k, [**float**]
) # *fin PT_LF_VMSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en ciaillement simple.

3.9.7 Classe PT_LF_VMAE

La classe **PT_LF_VMAE** permet de définir une loi de comportement de type Von Mises Avec Ecrouissage pour une fibre d'élément de poutre multi-fibre.

vmae = **PT_LF_VMAE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **K** = k, [**float**]
♦ **H** = h, [**float**]
) # *fin PT_LF_VMAE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en cisaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.

3.9.8 Classe PT_LF_CP

La classe **PT_LF_CP** permet de définir une loi de comportement de type Critère Parabolique pour une fibre d'élément de poutre multi-fibre.

cp = **PT_LF_CP**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **RC** = rc, [**float**]
- **RT** = rt, [**float**]

) # *fin PT_LF_CP*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple.

3.9.9 Classe PT_LF_WWS

La classe **PT_LF_WWS** permet de définir une loi de comportement de type Willam-Warnke Standard pour une fibre d'élément de poutre multi-fibre.

wws = **PT_LF_WWS**(
- **RO** = ro, [**float**]
- **YOUNG** = young, [**float**]
- **POISS** = poiss, [**float**]
- **FC** = fc, [**float**]
- **FT** = ft, [**float**]
- **FBC** = fbc, [**float**]
- **A0** = a0, [**float**]
- **B0** = b0, [**float**]
- **KAPPA** = kappa, [**float**]

) # *fin PT_LF_WWS*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.

3.9.10 Classe PT_LF_WWM

La classe **PT_LF_WWM** permet de définir une loi de comportement de type Willam-Warnke Modifié pour une fibre d'élément de poutre multi-fibre.

wwm = **PT_LF_WWM**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **FC** = fc, [**float**]
- ♦ **FT** = ft, [**float**]
- ♦ **FBC** = fbc, [**float**]
- ♦ **SIG** = sig, [**list**<**float**>]
- ♦ **A0** = a0, [**float**]
- ♦ **B0** = b0, [**float**]
- ♦ **KAPPA** = kappa, [**float**]

) # *fin PT_LF_WWM*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
SIG : liste de 3 réels donnant les valeurs des contraintes triaxiales de ruine.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.

3.9.11 Classe PT_CF

La classe **PT_CF** permet de définir les caractéristiques d'une fibre d'élément de poutre multi-fibre.

eli = **PT_CF**(
- ♦ **SF** = sf, [**float**]
- ♦ **X2** = x2, [**float**]
- ♦ **X3** = x3, [**float**]
- ♦ **ILOI** = iloi, [**float**]

) # *fin PT_CF*

SF : réel donnant la valeur de l'aire de la section d'une fibre ;
X2 : réel donnant la valeur de la coordonnée suivant l'axe 2 du centre de la fibre.
X3 : réel donnant la valeur de la coordonnée suivant l'axe 3 du centre de la fibre.
ILOI : entier donnant le rang de la loi de comportement à affecter à la fibre.

3.10 Données relatives au modèle de coque

3.10.1 Classe CO

La classe **CO** permet de définir les caractéristiques d'un groupe d'éléments de coque standard ou multi-couches.

```
co = CO(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                  ♦ IMOD = / 1,
                               ♦ CARA = cara, [CO_STD_ELI]
                             / 2,
                               ♦ CARA = cara, [CO_MC]
                  ♦ IMP = / 11,
                           / 12,
                           / 13,
                           / 14,
                           / 21,
                           / 22,
                           / 23,
                           / 24,
                  ♦ NNELG = nnelg, [int]
                / 'I',
      ) # fin CO
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**.
CARA : objet de type à choisir parmi :
- **CO_STD_ELI** pour un modèle de coque standard en élasticité linéaire isotrope,
- **CO_MC** pour un modèle de coque multicouche,

en cohérence avec le choix effectué pour **IMOD**.
IMP : entier indiquant le niveau d'impression des résultats à choisir parmi **11**, **12**, **13**, **14**, **21**, **22**, **23**, **24**.
NNELG : entier donnant le nombre maximum de noeuds d'un élément du groupe.

3.10.2 Classe CO_STD_ELI

La classe **CO_STD_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de coque.

```
eli = CO_STD_ELI(
      ♦ RO = ro, [float]
      ♦ YOUNG = young, [float]
      ♦ POISS = poiss, [float]
      ♦ EP = ep, [float]
      ) # fin CO_STD_ELI
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
EP : réel donnant l'épaisseur de la structure.

3.10.3 Classe CO_MC

La classe **CO_MC** permet de définir la géométrie et les matériaux d'une coque multi-couches.

```
mc = CO_MC(
    ♦ NCOU = ncou, [int]
    ♦ NLOI = nloi, [int]
    ♦ LOIS = lois, [list<CO_LC>]
    ♦ ILOI = iloi, [list<int>]
    ♦ EP = ep, [list<float>]
    ♦ EXC = exc, [list<float>]
    ) # fin CO_MC
```

NCOU : entier donnant le nombre de couches.
NLOI : entier donnant le nombre de lois de comportement différentes.
LOIS : liste d'objets de type **CO_LC** donnant les **NLOI** lois de comportements.
ILOI : liste d'entiers de taille **NCOU** affectant les lois (via leur rang dans **LOIS**) aux couches.
EP : liste de réels donnant les **NCOU** épaisseurs des couches.
EXC : liste de réels donnant les **NCOU** excentricités des couches.

3.10.4 Classe CO_LC

La classe **CO_LC** permet d'affecter une loi de comportement sur une couche d'élément de coque multi-couches.

```
lc = CO_LC(
    ♦ IMODC = / 1,
                ♦ CARA = cara, [CO_LC_ELI]
              / 11,
                ♦ CARA = cara, [CO_LC_VMSE]
              / 12,
                ♦ CARA = cara, [CO_LC_VMAE]
              / 15,
                ♦ CARA = cara, [CO_LC_CP]
              / 47,
                ♦ CARA = cara, [CO_LC_WWS]
              / 48,
                ♦ CARA = cara, [CO_LC_WWM]
    ) # fin CO_LC
```

IMODC : entier donnant le type de modèle mécanique utilisé pour la couche à choisir parmi **1**, **11**, **12**, **15**, **47**, **48**.
CARA : objet de type à choisir parmi :

· **CO_LC_ELI** pour une loi de type élasticité linéaire isotrope,
· **CO_LC_VMSE** pour une loi de type Von Mises sans écrouissage,
· **CO_LC_VMAE** pour une loi de type Von Mises avec écrouissage,
· **CO_LC_CP** pour une loi de type critère parabolique,
· **CO_LC_WWS** pour une loi de type Willam-Warnke standard,
· **CO_LC_WWM** pour une loi de type Willam-Warnke modifié,
en cohérence avec le choix effectué pour **IMODC**.

3.10.5 Classe CO_LC_ELI

La classe **CO_LC_ELI** permet de définir une loi de comportement de type élasticité linéaire isotrope pour une couche d'élément de coque multi-couches.

eli = **CO_LC_ELI**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
) # *fin CO_LC_ELI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.

3.10.6 Classe CO_LC_VMSE

La classe **CO_LC_VMSE** permet de définir une loi de comportement de type Von Mises Sans Ecrouissage pour une couche d'élément de coque multi-couches.

vmse = **CO_LC_VMSE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **K** = k, [**float**]
) # *fin CO_LC_VMSE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en ciaillement simple.

3.10.7 Classe CO_LC_VMAE

La classe **CO_LC_VMAE** permet de définir une loi de comportement de type Von Mises Avec Ecrouissage pour une couche d'élément de coque multi-couches.

vmae = **CO_LC_VMAE**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]

♦ **POISS** = poiss, [**float**]
♦ **K** = k, [**float**]
♦ **H** = h, [**float**]
) # *fin CO_LC_VMAE*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
K : réel donnant la valeur de la résistance en cisaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.

3.10.8 Classe CO_LC_CP

La classe **CO_LC_CP** permet de définir une loi de comportement de type critère parabolique pour une couche d'élément de coque multi-couches.

cp = **CO_LC_CP**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **RC** = rc, [**float**]
♦ **RT** = rt, [**float**]
) # *fin CO_LC_CP*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple.

3.10.9 Classe CO_LC_WWS

La classe **CO_LC_WWS** permet de définir une loi de comportement de type Willam-Warnke Standard pour une couche d'élément de coque multi-couches.

wws = **CO_LC_WWS**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **FC** = fc, [**float**]
♦ **FT** = ft, [**float**]
♦ **FBC** = fbc, [**float**]
♦ **A0** = a0, [**float**]
♦ **B0** = b0, [**float**]
♦ **KAPPA** = kappa, [**float**]
) # *fin CO_LC_WWS*

RO : réel donnant la valeur de la masse volumique.

YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.

3.10.10 Classe CO_LC_WWM

La classe **CO_LC_WWM** permet de définir une loi de comportement de type Willam-Warnke Modifié pour une couche d'élément de coque multi-couches.

wwm = **CO_LC_WWM**(
♦ **RO** = ro, [**float**]
♦ **YOUNG** = young, [**float**]
♦ **POISS** = poiss, [**float**]
♦ **FC** = fc, [**float**]
♦ **FT** = ft, [**float**]
♦ **FBC** = fbc, [**float**]
♦ **SIG** = sig, [**list**<**float**>]
♦ **A0** = a0, [**float**]
♦ **B0** = b0, [**float**]
♦ **KAPPA** = kappa, [**float**]
) # *fin CO_LC_WWM*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.
FC : réel donnant la valeur de la résistance en compression.
FT : réel donnant la valeur de la résistance en traction.
FBC : réel donnant la valeur de la résistance en compression biaxiale.
SIG : liste de 3 réels donnant les valeurs des contraintes triaxiales de ruine.
A0 : réel donnant la valeur de la limite d'élasticité initiale.
B0 : réel donnant la valeur de la limite d'élasticité finale.
KAPPA : réel donnant la valeur du facteur exponentiel d'écrouissage.

3.11 Données relatives au modèle de contact

3.11.1 Classe FD

La classe **FD** permet de définir les caractéristiques d'un groupe d'éléments de contact.

fd = **FD**(
♦ **NOMG** = nomg, [**str**]
♦ **ACTI** = / **'A'**,
♦ **ITAC** = / **1**,
/ **0**,

```
                ♦ IMOD = / 1,
                           ♦ CARA = cara, [FD_AD]
                         / 2,
                           ♦ CARA = cara, [FD_FC]
                         / 3,
                           ♦ CARA = cara, [FD_GP]
              / 'I',
) # fin FD
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
ITAC : entier indiquant si les éléments du groupe sont initialement en contact (**1**) ou non (**0**).
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **3**.
CARA : objet de type à choisir parmi :
- **FD_AD** pour une loi de type adhérence,
- **FD_FC** pour une loi de type frottement de Coulomb,
- **FD_GP** pour une loi de type glissement parfait,

en cohérence avec le choix effectué pour **IMOD**.

3.11.2 Classe FD_AD

La classe **FD_AD** permet d'affecter une loi de comportement de type adhérence aux éléments de contacts.

```
ad = FD_AD(
    ♦ COERI = coeri, [float]
    ♦ INAT = / 0,
             / 1,
             / 2,
             / 3,
               ♦ EP = ep, [float]
) # fin FD_AD
```

COERI : réel donnant la valeur du coefficient de rigidité du matériau fictif de contact.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant la valeur de l'épaisseur de la structure.

3.11.3 Classe FD_FC

La classe **FD_FC** permet d'affecter une loi de comportement de type frottement de coulomb aux éléments de contacts.

```
fc = FD_FC(
```

♦ **COERI** = coeri, [**float**]
♦ **RT** = rt, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **INAT** = / **0**,
/ **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin FD_FC*

COERI : réel donnant la valeur du coefficient de rigidité du matériau fictif de contact.
RT : réel donnant la valeur de la résistance à la traction.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement.
PSI : réel donnant la valeur de l'angle de dilatance.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant la valeur de l'épaisseur de la structure.

3.11.4 Classe FD_GP

La classe **FD_GP** permet d'affecter une loi de comportement de type glissement parfait aux éléments de contacts.

gp = **FD_GP**(
♦ **COERI** = coeri, [**float**]
♦ **RT** = rt, [**float**]
♦ **INAT** = / **0**,
/ **1**,
/ **2**,
/ **3**,
♦ **EP** = ep, [**float**]
) # *fin FD_GP*

COERI : réel donnant la valeur du coefficient de rigidité du matériau fictif de contact.
RT : réel donnant la valeur de la résistance à la traction.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème en déformation plane,
— **2** : pour un problème en déformation axisymétrique,
— **3** : pour un problème en contrainte plane.
EP : réel donnant la valeur de l'épaisseur de la structure.

3.12 Données relatives au modèle de joint

3.12.1 Classe EJ

La classe **EJ** permet de définir les caractéristiques d'un groupe d'éléments de joint. Elle est analogue à la classe **FD** (modèle de contact).

```
ej = EJ(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                 ♦ ITAC = / 1,
                          / 0,
                 ♦ IMOD = / 1,
                            ♦ CARA = cara, [EJ_AD]
                          / 2,
                            ♦ CARA = cara, [EJ_FC]
                          / 3,
                            ♦ CARA = cara, [EJ_GP]
               / 'I',
      ) # fin EJ
```

3.12.2 Classe EJ_AD

La classe **EJ_AD** permet d'affecter une loi de comportement de type adhérence aux éléments de joint. Elle est analogue à la classe **FD_AD**. Néanmoins, elle n'est pas utilisable en contrainte plane.

```
ad = EJ_AD(
      ♦ COERI = coeri, [float]
      ♦ INAT = / 0,
               / 1,
               / 2,
      ) # fin EJ_AD
```

3.12.3 Classe EJ_FC

La classe **EJ_FC** permet d'affecter une loi de comportement de type frottement de coulomb aux éléments de joint. Elle est analogue à la classe **FD_FC**. Néanmoins, elle n'est pas utilisable en contrainte plane.

```
fc = EJ_FC(
      ♦ COERI = coeri, [float]
      ♦ RT = rt, [float]
      ♦ C = c, [float]
      ♦ PHI = phi, [float]
      ♦ PSI = psi, [float]
      ♦ INAT = / 0,
               / 1,
```

/ **2**,
) # *fin EJ_FC*

3.12.4 Classe EJ_GP

La classe **EJ_GP** permet d'affecter une loi de comportement de type glissement parfait aux éléments de joint. Elle est analogue à la classe **EJ_GP**. Néanmoins, elle n'est pas utilisable en contrainte plane.

gp = **EJ_GP**(
♦ **COERI** = coeri, [**float**]
♦ **RT** = rt, [**float**]
♦ **INAT** = / **0**,
/ **1**,
/ **2**,
) # *fin EJ_GP*

3.12.5 Classe EJ_EL

La classe **EJ_EL** permet d'affecter une loi de comportement de type élasticité linéaire aux éléments de joint.

el = **EJ_EL**(
♦ **KNN** = knn, [**float**]
♦ **KTT** = ktt, [**float**]
) # *fin EJ_EL*

KNN : réel donnant la valeur de la raideur normale du joint.
KTT : réel donnant la valeur de la raideur tangentielle et transverse du joint.

3.12.6 Classe EJ_EPP

La classe **EJ_EPP** permet d'affecter une loi de comportement de type élastique parfaitement plastique aux éléments de joint.

epp = **EJ_EPP**(
♦ **KNN** = knn, [**float**]
♦ **KTT** = ktt, [**float**]
♦ **RT** = rt, [**float**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
) # *fin EJ_EPP*

KNN : réel donnant la valeur de la raideur normale du joint.
KTT : réel donnant la valeur de la raideur tangentielle et transverse du joint.

RC : réel donnant la valeur maximale de la contrainte normale.
C : réel donnant la valeur de la cohésion du joint.
PHI : réel donnant la valeur de l'angle de frottement (en degrés).
PSI : réel donnant la valeur de l'angle de dilatance (en degrés).

3.13 Données relatives au modèle de barre bidimensionnelle

3.13.1 Classe BB

La classe **BB** permet de définir les caractéristiques d'un groupe d'éléments de barre bidimensionnelle.

```
bb = BB(
    ♦ NOMG = nomg, [str]
    ♦ ACTI = / 'A',
                ♦ IMOD = / 1,
                            ♦ CARA = cara, [BB_ELI]
             / 'I',
    ) # fin BB
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**.
CARA : objet de type à choisir parmi :
- **BB_ELI** pour une loi de type élasticite linéaire isotrope,

en cohérence avec le choix effectué pour **IMOD**.

3.13.2 Classe BB_ELI

La classe **BB_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de barre bidimensionnelle.

```
eli = BB_ELI(
    ♦ YOUNG = young, [float]
    ♦ S = s, [float]
    ) # fin BB_ELI
```

YOUNG : réel donnant la valeur du module d'Young.
S : réel donnant la valeur de la section de la barre.

3.14 Données relatives au modèle de barre tridimensionnelle

3.14.1 Classe BT

La classe **BT** permet de définir les caractéristiques d'un groupe d'éléments de barre tridimensionnelle.

```
bt = BT(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                  ♦ IMOD = / 1,
                              ♦ CARA = cara, [BT_ELI]
               / 'I',
      ) # fin BT
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**.
CARA : objet de type à choisir parmi :
· **BT_ELI** pour une loi de type élasticite linéaire isotrope,
en cohérence avec le choix effectué pour **IMOD**.

3.14.2 Classe BT_ELI

La classe **BT_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de barre tridimensionnelle.

```
eli = BT_ELI(
      ♦ YOUNG = young, [float]
      ♦ S = s, [float]
      ) # fin BT_ELI
```

YOUNG : réel donnant la valeur du module d'Young.
S : réel donnant la valeur de la section de la barre.

3.15 Données relatives au modèle de barre frottante

3.15.1 Classe KR

La classe **KR** permet de définir les caractéristiques d'un groupe d'éléments de barre (bidimensionnelle ou tridimensionnelle) frottante.

```
kr = KR(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                  ♦ IMOD = / 1,
                              ♦ CARA = cara, [KR_ELI]
               / 'I',
      ) # fin KR
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.

IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**.
CARA : objet de type à choisir parmi :
· **KR_ELI** pour une loi de type élasticite linéaire isotrope,
en cohérence avec le choix effectué pour **IMOD**.

3.15.2 Classe KR_ELI

La classe **KR_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de barre frottante.

```
eli = KR_ELI(
    ♦ YOUNG = young, [float]
    ♦ S = s, [float]
    ♦ ICINT = / 1,
                 ♦ CARAI = carai, [KR_INT_EL]
              / 4,
                 ♦ CARAI = carai, [KR_INT_ELP]
              / 14,
                 ♦ CARAI = carai, [KR_INT_EBP]
              / 15,
                 ♦ CARAI = carai, [KR_INT_ERP]
    ) # fin KR_ELI
```

YOUNG : réel donnant la valeur du module d'Young.
S : réel donnant la valeur de la section de la barre.
ICINT : entier indiquant le type de modèle mécanique utilisé pour l'intéraction entre la barre et le milieu qui l'entoure :
— **1** : élasticité linéaire,
— **4** : élasticité linéaire et plasticité parfaite,
— **14** : élasticité bilinéaire et plasticité parfaite,
— **15** : élasticité loi racine et plasticité parfaite.
CARAI : objet définissant les paramètres du modèle d'intéraction à choisir parmi :
· **KR_INT_EL**,
· **KR_INT_ELP**,
· **KR_INT_EBP**,
· **KR_INT_ERP**.
en cohérence avec le choix effectué pour **ICINT**.

3.15.3 Classe KR_INT_EL

La classe **KR_INT_EL** permet de définir les paramètres du modèle d'intéraction entre la barre et le milieu qui l'entoure de type élasticité linéaire.

```
el = KR_INT_EL(
    ♦ CI = ci, [float]
    ) # fin KR_INT_EL
```

CI : réel donnant la valeur du coefficient d'intéraction.

3.15.4 Classe KR_INT_ELP

La classe **KR_INT_ELP** permet de définir les paramètres du modèle d'intéraction entre la barre et le milieu qui l'entoure de type élasticité linéaire et plasticité parfaite.

elp = **KR_INT_ELP**(
♦ **CI** = ci, [**float**]
♦ **IMAX** = imax, [**float**]
) # *fin KR_INT_ELP*

CI : réel donnant la valeur du coefficient d'intéraction,
IMAX : réel donnant la valeur maximale de la force linéique d'intéraction.

3.15.5 Classe KR_INT_EBP

La classe **KR_INT_EBP** permet de définir les paramètres du modèle d'intéraction entre la barre et le milieu qui l'entoure de type élasticité bilinéaire et plasticité parfaite.

ebp = **KR_INT_EBP**(
♦ **CI1** = ci1, [**float**]
♦ **CI2** = ci2, [**float**]
♦ **I1** = i1, [**float**]
♦ **I2** = i2, [**float**]
) # *fin KR_INT_EBP*

CI1 : réel donnant la valeur du coefficient d'intéraction initial,
CI2 : réel donnant la valeur du coefficient d'intéraction secondaire,
I1 : réel donnant la valeur seuil de la force linéique d'intéraction,
I2 : réel donnant la valeur maximale de la force linéique d'intéraction.

3.15.6 Classe KR_INT_ERP

La classe **KR_INT_ERP** permet de définir les paramètres du modèle d'intéraction entre la barre et le milieu qui l'entoure de type élasticité bilinéaire et plasticité parfaite.

erp = **KR_INT_ERP**(
♦ **CI** = ci, [**float**]
♦ **DELTAREF** = deltaref, [**float**]
♦ **IMAX** = imax, [**float**]
) # *fin KR_INT_ERP*

CI : réel donnant la valeur du coefficient d'intéraction initial,
DELTAREF : réel donnant la valeur du déplacement relatif pour lequel la force d'intéraction atteint la valeur maximale,
IMAX : réel donnant la valeur maximale de la force linéique d'intéraction.

3.16 Données relatives aux relations linéaires

3.16.1 Classe RL

La classe **RL** permet de définir des relations linéaires.

```
rl = RL(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                  ♦ IDL = idl, [list<int>]
                  ♦ C = c, [list<float>]
                  ♦ P = p, [float]
               / 'I',
      ) # fin RL
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IDL : liste d'entiers donnant les numéros (entre 1 et 6) des degrés de liberté intervenant dans la relation linéaire.
C : liste de réels de même taille que **IDL** donnant les coefficients de la relation linéaire.
P : réel donnant le facteur de pénalisation utilisé pour imposer la relation linéaire.

3.17 Données relatives aux éléments spéciaux

3.17.1 Classe SP

La classe **SP** permet de définir des caractéristiques d'éléments spéciaux.

```
sp = SP(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                  ♦ MFICH = / 0,
                              ♦ RE = re, [list<float>]
                            / 1,
                              ♦ NOMF = nomf, [str]
               / 'I',
      ) # fin SP
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
MFICH : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier pour la lecture de la matrice élémentaire.
RE : liste de réels donnant les termes de la matrice élémentaire rangés par colonnes decendantes.
NOMF : chaîne de caractères donnant le nom du fichier sur lequel est lue la matrice élémentaire.

3.18 Données relatives au modèle de mécanique bidimensionnelle axisymétrique

3.18.1 Classe AX

La classe **AX** permet de définir les caractéristiques d'un groupe d'éléments de mécanique bidimensionnelle pour les structures à géométrie de révolution soumises à un chargement quelconque.

```
ax = AX(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                 ♦ IMOD = / 1,
                            ♦ CARA = cara, [AX_ELI]
                 ♦ NHAR = nhar, [int]
                 ♦ ISYM = / -1,
                          / 0,
                          / 1,
               / 'I',
      ) # fin AX
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**.
CARA : objet de type à choisir parmi :
- **AX_ELI** pour une loi de type élasticite linéaire isotrope,

en cohérence avec le choix effectué pour **IMOD**.
NHAR : entier donnant le numéro d'ordre de l'harmonique considérée pour une utilisation avec un module de calcul autre que AXIF, ou à fixer à la valeur **99** pour une utilisation avec le module AXIF.
ISYM : entier valant **0** si **NHAR** = 0, ou bien indiquant le caractère symétrique (**1**) ou antisymétrique (**-1**) de l'harmonique sinon.

3.18.2 Classe AX_ELI

La classe **AX_ELI** permet d'affecter une loi de comportement de type élasticité linéaire isotrope aux éléments de mécanique bidimensionnelle axisymétrique.

```
eli = AX_ELI(
      ♦ RO = ro, [float]
      ♦ YOUNG = young, [float]
      ♦ POISS = poiss, [float]
      ) # fin AX_ELI
```

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young.
POISS : réel donnant la valeur du coefficient de Poisson.

3.19 Données relatives au modèle de diffusion bidimensionnelle

3.19.1 Classe DB

La classe **DB** permet de définir les caractéristiques d'un groupe d'éléments de diffusion bidimensionnelle.

```
db = DB(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                 ♦ IMOD = / 1,
                            ♦ CARA = cara, [DB_CCH]
                          / 2,
                            ♦ CARA = cara, [DB_EMP]
                          / 3,
                            ♦ CARA = cara, [DB_NLG]
                          / 4,
                            ♦ CARA = cara, [DB_CP]
                          / 40,
                            ♦ CARA = cara, [DB_CPP]
                 ♦ INAT = / 1,
                          / 2,
               / 'I',
) # fin DB
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **3**, **4**, **40**.
CARA : objet de type à choisir parmi :

- **DB_CCH** pour un modèle de type conduction de la chaleur
- **DB_EMP** pour un modèle de type écoulement en milieu poreux
- **DB_NLG** pour un modèle de type non-linéaire général
- **DB_CP** pour un modèle de type écoulement en milieu poreux non saturé avec utilisation de courbes préprogrammées
- **DB_CPP** pour un modèle de type écoulement en milieu poreux non saturé avec données de courbes point par point

en cohérence avec le choix effectué pour **IMOD**.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème plan,
— **2** : pour un problème axisymétrique.

3.19.2 Classe DB_CCH

La classe **DB_CCH** permet d'affecter des caractéristiques de conduction de la chaleur aux éléments de diffusion bidimensionnelle.

```
cch = DB_CCH(
      ♦ AKX = akx, [float]
```

♦ **AKY** = aky, [**float**]
♦ **AKXY** = akxy, [**float**]
♦ **CC** = cc, [**float**]
) # *fin DB_CCH*

AKX : réel donnant la valeur du coefficient AK_x du tenseur de conductivité.
AKY : réel donnant la valeur du coefficient AK_y du tenseur de conductivité.
AKXY : réel donnant la valeur du coefficient AK_{xy} du tenseur de conductivité.
CC : réel donnant la valeur de la capacité calorifique.

3.19.3 Classe DB_EMP

La classe **DB_EMP** permet d'affecter des caractéristiques d'écoulement en milieu poreux aux éléments de diffusion bidimensionnelle.

emp = **DB_EMP**(
♦ **AKX** = akx, [**float**]
♦ **AKY** = aky, [**float**]
♦ **AKXY** = akxy, [**float**]
♦ **CE** = ce, [**float**]
) # *fin DB_EMP*

AKX : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité.
AKY : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité.
AKXY : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité.
CE : réel donnant la valeur du coefficient d'emmagasinement.

3.19.4 Classe DB_NLG

La classe **DB_NLG** permet d'affecter des caractéristiques de modèle non linéaire général aux éléments de diffusion bidimensionnelle.

nlg = **DB_NLG**(
♦ **AKX** = akx, [**float**]
♦ **AKY** = aky, [**float**]
♦ **AKXY** = akxy, [**float**]
♦ **CE** = ce, [**float**]
♦ **NC** = nc, [**int**]
♦ **VCOURB** = vcourb, [**list**<**float**>]
) # *fin DB_NLG*

AKX : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité (resp. conductivité).
AKY : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité (resp. conductivité).
AKXY : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité (resp. conductivité).

CE : réel donnant la valeur du coefficient d'emmagasinement (resp. capacité calorifique).
NC : entier (≥ 4) donnant le nombre de triplets de valeurs définissant les courbes.
VCOURB : liste de réels donnant les triplets de valeurs $(u, K_r(u), C_r(u))$ définissant les courbes de la perméabilité (resp. conductivité) relative K_r et du coefficient d'emmagasinement (resp. capacité calorifique) relatif C_r en fonction de la pression (resp. température) u.

3.19.5 Classe DB_CP

La classe **DB_CP** permet d'affecter des caractéristiques d'écoulement en milieu poreux non saturé avec courbes préprogrammées aux éléments de diffusion bidimensionnelle.

cp = **DB_CP**(
- **AKXS** = akxs, [**float**]
- **AKYS** = akys, [**float**]
- **AKXYS** = akxys, [**float**]
- **CE** = ce, [**float**]
- **A** = a, [**float**]
- **B** = b, [**float**]
- **C** = c, [**float**]
- **D** = d, [**float**]
- **CER** = cer, [**float**]
- **CES** = ces, [**float**]
- **P0** = p0, [**float**]

) # *fin DB_CP*

AKXS : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité à saturation.
AKYS : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité à saturation.
AKXYS : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité à saturation.
CE : réel donnant la valeur de la porosité efficace.
A : réel donnant la valeur du coefficient A de la loi de perméabilité relative dans le domaine non saturé.
B : réel donnant la valeur du coefficient B de la loi de perméabilité relative dans le domaine non saturé.
C : réel donnant la valeur du coefficient C de la loi de teneur en eau relative dans le domaine non saturé.
D : réel donnant la valeur du coefficient D de la loi de teneur en eau relative dans le domaine non saturé.
CER : réel donnant la valeur du coefficient d'emmagasinement résiduel dans le domaine non saturé.
CES : réel donnant la valeur du coefficient d'emmagasinement dans le domaine saturé.
P0 : réel donnant la valeur de la pression de référence.

3.19.6 Classe DB_CPP

La classe **DB_CPP** permet d'affecter des caractéristiques d'écoulement en milieu poreux non saturé avec courbes définies point par point aux éléments de diffusion bidimensionnelle.

```
cpp = DB_CPP(
      ♦ AKXS = akxs, [float]
      ♦ AKYS = akys, [float]
      ♦ AKXYS = akxys, [float]
      ♦ CE = ce, [float]
      ♦ NC = nc, [int]
      ♦ VCOURB = vcourb, [list<float>]
      ♦ CES = ces, [float]
      ) # fin DB_CPP
```

AKXS : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité à saturation.
AKYS : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité à saturation.
AKXYS : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité à saturation.
CE : réel donnant la valeur de la porosité efficace.
NC : entier (≥ 4) donnant le nombre de triplets de valeurs définissant les courbes.
VCOURB : liste de réels donnant les triplets de valeurs $(P, K_r(P), \theta_r(P))$ définissant les courbes de la perméabilité relative K_r et de la teneur en eau relative θ_r en fonction de la pression P.
CES : réel donnant la valeur du coefficient d'emmagasinement dans le domaine saturé.

3.20 Données relatives au modèle de diffusion tridimensionnelle

3.20.1 Classe DT

La classe **DT** permet de définir les caractéristiques d'un groupe d'éléments de diffusion tridimensionnelle.

```
dt = DT(
     ♦ NOMG = nomg, [str]
     ♦ ACTI = / 'A',
                ♦ IMOD = / 1,
                           ♦ CARA = cara, [DT_CCH]
                         / 2,
                           ♦ CARA = cara, [DT_EMP]
                         / 3,
                           ♦ CARA = cara, [DT_NLG]
                         / 4,
                           ♦ CARA = cara, [DT_CP]
                         / 40,
                           ♦ CARA = cara, [DT_CPP]
              / 'I',
     ) # fin DT
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **3**, **4**, **40**.

CARA : objet de type à choisir parmi :
- **DT_CCH** pour un modèle de type conduction de la chaleur
- **DT_EMP** pour un modèle de type écoulement en milieu poreux
- **DT_NLG** pour un modèle de type non-linéaire général
- **DT_CP** pour un modèle de type écoulement en milieu poreux non saturé avec utilisation de courbes préprogrammées
- **DT_CPP** pour un modèle de type écoulement en milieu poreux non saturé avec données de courbes point par point

en cohérence avec le choix effectué pour **IMOD**.

3.20.2 Classe DT_CCH

La classe **DT_CCH** permet d'affecter des caractéristiques de conduction de la chaleur aux éléments de diffusion tridimensionnelle.

cch = **DT_CCH**(
- ♦ **AKX** = akx, [**float**]
- ♦ **AKY** = aky, [**float**]
- ♦ **AKZ** = akz, [**float**]
- ♦ **AKXY** = akxy, [**float**]
- ♦ **AKYZ** = akyz, [**float**]
- ♦ **AKXZ** = akxz, [**float**]
- ♦ **CC** = cc, [**float**]

) # *fin DT_CCH*

AKX : réel donnant la valeur du coefficient AK_x du tenseur de conductivité.
AKY : réel donnant la valeur du coefficient AK_y du tenseur de conductivité.
AKZ : réel donnant la valeur du coefficient AK_z du tenseur de conductivité.
AKXY : réel donnant la valeur du coefficient AK_{xy} du tenseur de conductivité.
AKYZ : réel donnant la valeur du coefficient AK_{yz} du tenseur de conductivité.
AKXZ : réel donnant la valeur du coefficient AK_{xz} du tenseur de conductivité.
CC : réel donnant la valeur de la capacité calorifique.

3.20.3 Classe DT_EMP

La classe **DT_EMP** permet d'affecter des caractéristiques d'écoulement en milieu poreux aux éléments de diffusion tridimensionnelle.

emp = **DT_EMP**(
- ♦ **AKX** = akx, [**float**]
- ♦ **AKY** = aky, [**float**]
- ♦ **AKZ** = akz, [**float**]
- ♦ **AKXY** = akxy, [**float**]
- ♦ **AKYZ** = akyz, [**float**]
- ♦ **AKXZ** = akxz, [**float**]
- ♦ **CE** = ce, [**float**]

) # *fin DT_EMP*

AKX : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité.

AKY : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité.
AKZ : réel donnant la valeur du coefficient AK_z du tenseur de perméabilité.
AKXY : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité.
AKYZ : réel donnant la valeur du coefficient AK_{yz} du tenseur de perméabilité.
AKXZ : réel donnant la valeur du coefficient AK_{xz} du tenseur de perméabilité.
CE : réel donnant la valeur du coefficient d'emmagasinement.

3.20.4 Classe DT_NLG

La classe **DT_NLG** permet d'affecter des caractéristiques de modèle non linéaire général aux éléments de diffusion tridimensionnelle.

nlg = **DT_NLG**(
- **AKX** = akx, [**float**]
- **AKY** = aky, [**float**]
- **AKZ** = akz, [**float**]
- **AKXY** = akxy, [**float**]
- **AKYZ** = akyz, [**float**]
- **AKXZ** = akxz, [**float**]
- **CE** = ce, [**float**]
- **NC** = nc, [**int**]
- **VCOURB** = vcourb, [**list**<**float**>]

) # *fin DT_NLG*

AKX : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité (resp. conductivité).
AKY : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité (resp. conductivité).
AKZ : réel donnant la valeur du coefficient AK_z du tenseur de perméabilité (resp. conductivité).
AKXY : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité (resp. conductivité).
AKYZ : réel donnant la valeur du coefficient AK_{yz} du tenseur de perméabilité (resp. conductivité).
AKXZ : réel donnant la valeur du coefficient AK_{xz} du tenseur de perméabilité (resp. conductivité).
CE : réel donnant la valeur du coefficient d'emmagasinement (resp. capacité calorifique).
NC : entier (≥ 4) donnant le nombre de triplets de valeurs définissant les courbes.
VCOURB : liste de réels donnant les triplets de valeurs $(u, K_r(u), C_r(u))$ définissant les courbes de la perméabilité (resp. conductivité) relative K_r et du coefficient d'emmagasinement (resp. capacité calorifique) relatif C_r en fonction de la pression (resp. température) u.

3.20.5 Classe DT_CP

La classe **DT_CP** permet d'affecter des caractéristiques d'écoulement en milieu poreux non saturé avec courbes préprogrammées aux éléments de diffusion tridimensionnelle.

cp = **DT_CP**(

♦ **AKXS** = akxs, [**float**]
♦ **AKYS** = akys, [**float**]
♦ **AKZS** = akzs, [**float**]
♦ **AKXYS** = akxys, [**float**]
♦ **AKYZS** = akyzs, [**float**]
♦ **AKXZS** = akxzs, [**float**]
♦ **CE** = ce, [**float**]
♦ **A** = a, [**float**]
♦ **B** = b, [**float**]
♦ **C** = c, [**float**]
♦ **D** = d, [**float**]
♦ **CER** = cer, [**float**]
♦ **CES** = ces, [**float**]
♦ **P0** = p0, [**float**]
) # *fin DT_CP*

AKXS : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité à saturation.
AKYS : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité à saturation.
AKZS : réel donnant la valeur du coefficient AK_z du tenseur de perméabilité à saturation.
AKXYS : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité à saturation.
AKYZS : réel donnant la valeur du coefficient AK_{yz} du tenseur de perméabilité à saturation.
AKXZS : réel donnant la valeur du coefficient AK_{xz} du tenseur de perméabilité à saturation.
CE : réel donnant la valeur de la porosité efficace.
A : réel donnant la valeur du coefficient A de la loi de perméabilité relative dans le domaine non saturé.
B : réel donnant la valeur du coefficient B de la loi de perméabilité relative dans le domaine non saturé.
C : réel donnant la valeur du coefficient C de la loi de teneur en eau relative dans le domaine non saturé.
D : réel donnant la valeur du coefficient D de la loi de teneur en eau relative dans le domaine non saturé.
CER : réel donnant la valeur du coefficient d'emmagasinement résiduel dans le domaine non saturé.
CES : réel donnant la valeur du coefficient d'emmagasinement dans le domaine saturé.
P0 : réel donnant la valeur de la pression de référence.

3.20.6 Classe DT_CPP

La classe **DT_CPP** permet d'affecter des caractéristiques d'écoulement en milieu poreux non saturé avec courbes définies point par point aux éléments de diffusion tridimensionnelle.

cpp = **DT_CPP**(
♦ **AKXS** = akxs, [**float**]
♦ **AKYS** = akys, [**float**]
♦ **AKZS** = akzs, [**float**]
♦ **AKXYS** = akxys, [**float**]

♦ **AKYZS** = akyzs, [**float**]
♦ **AKXZS** = akxzs, [**float**]
♦ **CE** = ce, [**float**]
♦ **NC** = nc, [**int**]
♦ **VCOURB** = vcourb, [**list**<**float**>]
♦ **CES** = ces, [**float**]
) # *fin DT_CPP*

AKXS : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité à saturation.
AKYS : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité à saturation.
AKZS : réel donnant la valeur du coefficient AK_z du tenseur de perméabilité à saturation.
AKXYS : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité à saturation.
AKYZS : réel donnant la valeur du coefficient AK_{yz} du tenseur de perméabilité à saturation.
AKXZS : réel donnant la valeur du coefficient AK_{xz} du tenseur de perméabilité à saturation.
CE : réel donnant la valeur de la porosité efficace.
NC : entier (≥ 4) donnant le nombre de triplets de valeurs définissant les courbes.
VCOURB : liste de réels donnant les triplets de valeurs $(P, K_r(P), \theta_r(P))$ définissant les courbes de la perméabilité relative K_r et de la teneur en eau relative θ_r en fonction de la pression P.
CES : réel donnant la valeur du coefficient d'emmagasinement dans le domaine saturé.

3.21 Données relatives au modèle d'échange bidimensionnel

3.21.1 Classe EB

La classe **EB** permet de définir les caractéristiques d'un groupe d'éléments d'échange bidimensionnel.

```
eb = EB(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                  ♦ IMOD = / 1,
                               ♦ CARA = cara, [EB_CCH]
                             / 2,
                               ♦ CARA = cara, [EB_EMP]
                             / 3,
                               ♦ CARA = cara, [EB_NLG]
                  ♦ INAT = / 1,
                             / 2,
               / 'I',
      ) # fin EB
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.

IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **3**.
CARA : objet de type à choisir parmi :
- **EB_CCH** pour un modèle de type conduction de la chaleur
- **EB_EMP** pour un modèle de type écoulement en milieu poreux
- **EB_NLG** pour un modèle de type non-linéaire général

en cohérence avec le choix effectué pour **IMOD**.
INAT : entier indiquant la nature du problème étudié à choisir parmi :
— **1** : pour un problème plan,
— **2** : pour un problème axisymétrique.

3.21.2 Classe EB_CCH

La classe **EB_CCH** permet d'affecter des caractéristiques de conduction de la chaleur poreux aux éléments d'échange bidimensionnel.

cch = **EB_CCH**(
♦ **ECH** = ech, [**float**]
) # *fin EB_CCH*

ECH : réel donnant la valeur du coefficient d'échange.

3.21.3 Classe EB_EMP

La classe **EB_EMP** permet d'affecter des caractéristiques d'écoulement en milieu poreux aux éléments d'échange bidimensionnel.

emp = **EB_EMP**(
♦ **ECH** = ech, [**float**]
) # *fin EB_EMP*

ECH : réel donnant la valeur du coefficient d'échange.

3.21.4 Classe EB_NLG

La classe **EB_NLG** permet d'affecter des caractéristiques de modèle non linéaire général aux éléments d'échange bidimensionnel.

nlg = **EB_NLG**(
♦ **ECH** = ech, [**float**]
♦ **NC** = nc, [**int**]
♦ **VECH** = vech, [**list**<**float**>]
) # *fin EB_NLG*

ECH : réel donnant la valeur du coefficient d'échange.
NC : entier (≥ 2) donnant le nombre de triplets de valeurs définissant la courbe.
VECH : liste de réels donnant les couples de valeurs $(u, h_r(u))$ définissant la courbe du coefficient d'échange relatif h_r en fonction du paramètre u.

3.22 Données relatives au modèle d'échange tridimensionnel

3.22.1 Classe ET

La classe **ET** permet de définir les caractéristiques d'un groupe d'éléments d'échange tridimensionnel.

```
et = ET(
      ♦ NOMG = nomg, [str]
      ♦ ACTI = / 'A',
                 ♦ IMOD = / 1,
                            ♦ CARA = cara, [ET_CCH]
                          / 2,
                            ♦ CARA = cara, [ET_EMP]
                          / 3,
                            ♦ CARA = cara, [ET_NLG]
               / 'I',
      ) # fin ET
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **3**.
CARA : objet de type à choisir parmi :
- **ET_CCH** pour un modèle de type conduction de la chaleur
- **ET_EMP** pour un modèle de type écoulement en milieu poreux
- **ET_NLG** pour un modèle de type non-linéaire général

en cohérence avec le choix effectué pour **IMOD**.

3.22.2 Classe ET_CCH

La classe **ET_CCH** permet d'affecter des caractéristiques de conduction de la chaleur poreux aux éléments d'échange tridimensionnel.

```
cch = ET_CCH(
      ♦ ECH = ech, [float]
      ) # fin ET_CCH
```

ECH : réel donnant la valeur du coefficient d'échange.

3.22.3 Classe ET_EMP

La classe **ET_EMP** permet d'affecter des caractéristiques d'écoulement en milieu poreux aux éléments d'échange tridimensionnel.

```
emp = ET_EMP(
      ♦ ECH = ech, [float]
      ) # fin ET_EMP
```

ECH : réel donnant la valeur du coefficient d'échange.

3.22.4 Classe ET_NLG

La classe **ET_NLG** permet d'affecter des caractéristiques de modèle non linéaire général aux éléments d'échange tridimensionnel.

```
nlg = ET_NLG(
      ♦ ECH = ech, [float]
      ♦ NC = nc, [int]
      ♦ VECH = vech, [list<float>]
      ) # fin ET_NLG
```

ECH : réel donnant la valeur du coefficient d'échange.
NC : entier (≥ 2) donnant le nombre de triplets de valeurs définissant la courbe.
VECH : liste de réels donnant les couples de valeurs $(u, h_r(u))$ définissant la courbe du coefficient d'échange relatif h_r en fonction du paramètre u.

3.23 Données relatives au modèle bidimensionnel pour la recherche d'une surface libre

3.23.1 Classe SB

La classe **SB** permet de définir les caractéristiques d'un groupe d'éléments bidimensionnels discontinus pour la recherche d'une surface libre.

```
sb = SB(
     ♦ NOMG = nomg, [str]
     ♦ ACTI = / 'A',
                ♦ IMOD = / 1,
                         / 2,
                ♦ INAT = / 1,
                         / 2,
                ♦ AKXS = akxs, [float]
                ♦ AKYS = akys, [float]
                ♦ AKXYS = akxys, [float]
              / 'I',
     ) # fin SB
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi :
— **1** : si le groupe d'éléments n'est pas traversé par la surface libre,
— **2** : sinon.
AKXS : réel donnant la valeur du coefficient AK_x du tenseur de perméabilité à saturation.
AKYS : réel donnant la valeur du coefficient AK_y du tenseur de perméabilité à saturation.
AKXYS : réel donnant la valeur du coefficient AK_{xy} du tenseur de perméabilité à saturation.
INAT : entier indiquant la nature du problème étudié à choisir parmi :

— **1** : pour un problème plan,
— **2** : pour un problème axisymétrique.

3.24 Données relatives au modèle bidimensionnel de thermo-mécanique des milieux poreux

3.24.1 Classe OB

La classe **OB** permet de définir les caractéristiques d'un groupe d'éléments pour la thermo-mécanique bidimensionnelle des milieux poreux.

```
ob = OB(
    ♦ NOMG = nomg, [str]
    ♦ ACTI = / 'A',
                ♦ IMOD = / 1,
                           ♦ CARA = cara, [OB_ELI]
                         / 2,
                           ♦ CARA = cara, [OB_ELO]
                         / 10,
                           ♦ CARA = cara, [OB_MC]
                         / 11,
                           ♦ CARA = cara, [OB_VMSE]
                         / 12,
                           ♦ CARA = cara, [OB_VMAE]
                         / 13,
                           ♦ CARA = cara, [OB_DPSE]
                         / 14,
                           ♦ CARA = cara, [OB_DPAE]
                         / 15,
                           ♦ CARA = cara, [OB_CP]
                         / 16,
                           ♦ CARA = cara, [OB_VE]
                         / 17,
                           ♦ CARA = cara, [OB_NO]
                         / 18,
                           ♦ CARA = cara, [OB_CCM]
                         / 19,
                           ♦ CARA = cara, [OB_PH]
                         / 20,
                           ♦ CARA = cara, [OB_CO]
    ♦ INAT = / 1,
             / 2,
) # fin OB
```

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif (**'A'**) ou inactif (**'I'**) du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi **1**, **2**, **10**, **11**, **12**, **13**, **14**, **15**, **16**, **17**, **18**, **19**, **20**.

CARA : objet de type à choisir parmi :
- **OB_ELI** pour une loi de type thermo-poro-élasticité linéaire isotrope,
- **OB_ELO** pour une loi de type thermo-poro-élasticité linéaire orthotrope de révolution,
- **OB_MCSE** pour une loi thermo-poro-élastoplastique de type Mohr-Coulomb,
- **OB_VMSE** pour une loi thermo-poro-élastoplastique de type Von Mises sans écrouissage,
- **OB_VMAE** pour une loi thermo-poro-élastoplastique de type Von Mises avec écrouissage,
- **OB_DPSE** pour une loi thermo-poro-élastoplastique de type Drucker-Prager sans écrouissage,
- **OB_DPAE** pour une loi thermo-poro-élastoplastique de type Drucker-Prager avec écrouissage,
- **OB_CP** pour une loi thermo-poro-élastoplastique de type critère parabolique,
- **OB_VE** pour une loi thermo-poro-élastoplastique de type Vermeer,
- **OB_NO** pour une loi thermo-poro-élastoplastique de type Nova,
- **OB_CCM** pour une loi thermo-poro-élastoplastique de type Cam Clay modifié,
- **OB_PH** pour une loi thermo-poro-élastoplastique de type Prevost-Hoeg,
- **OB_CO** pour une loi thermo-poro-élastoplastique de type critère orienté,

en cohérence avec le choix effectué pour **IMOD**.

3.24.2 Classe OB_ELI

La classe **OB_ELI** permet d'affecter un comportement de type thermo-poro-élasticité linéaire isotrope aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

eli = **OB_ELI**(
- ♦ **RO** = ro, [**float**]
- ♦ **YOUNG** = young, [**float**]
- ♦ **POISS** = poiss, [**float**]
- ♦ **ROF** = rof, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **M** = m, [**float**]
- ♦ **PX** = px, [**float**]
- ♦ **PY** = py, [**float**]
- ♦ **PXY** = pxy, [**float**]
- ♦ **B** = b, [**float**]
- ♦ **CE0** = ce0, [**float**]
- ♦ **CX** = cx, [**float**]
- ♦ **CY** = cy, [**float**]
- ♦ **CXY** = cxy, [**float**]
- ♦ **A0** = a0, [**float**]
- ♦ **AMX3** = amx3, [**float**]
- ♦ **T0** = t0, [**float**]

) # *fin OB_ELI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young drainé.
POISS : réel donnant la valeur du coefficient de Poisson drainé.
ROF : réel donnant la valeur de la masse volumique du fluide.

PHI : réel donnant la valeur de la porosité.
M : réel donnant la valeur du module d'incompressibilité de Biot.
PX : réel donnant la valeur du coefficient de perméabilité suivant Ox.
PY : réel donnant la valeur du coefficient de perméabilité suivant Oy .
PXY : réel donnant la valeur du coefficient de perméabilité suivant Oxy.
B : réel donnant la valeur du coefficient de Biot.
CE0 : réel donnant la valeur de la chaleur volumique à déformations constantes drainées.
CX : réel donnant la valeur du coefficient de conductivité thermique suivant Ox.
CY : réel donnant la valeur du coefficient de conductivité thermique suivant Oy.
CXY : réel donnant la valeur du coefficient de conductivité thermique suivant Oxy.
A0 : réel donnant la valeur du coefficient thermoélastique de compressibilité à déformation volumique nulle et en condition drainée du matériau.
AMX3 : réel donnant la valeur du coefficient de dilatation à déformation volumique nulle en condition drainée.
T0 : réel donnant la valeur de la température de référence.

3.24.3 Classe OB_ELO

La classe **OB_ELO** permet d'affecter un comportement de type thermo-poro-élasticité linéaire orthotrope aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

elo = **OB_ELO**(
- ♦ **RO** = ro, [**float**]
- ♦ **E1** = e1, [**float**]
- ♦ **E2** = e2, [**float**]
- ♦ **P1** = p1, [**float**]
- ♦ **P2** = p2, [**float**]
- ♦ **G2** = g2, [**float**]
- ♦ **TETA** = teta, [**float**]
- ♦ **ROF** = rof, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **M** = m, [**float**]
- ♦ **PX** = px, [**float**]
- ♦ **PY** = py, [**float**]
- ♦ **PXY** = pxy, [**float**]
- ♦ **B1** = b1, [**float**]
- ♦ **B2** = b2, [**float**]
- ♦ **CE0** = ce0, [**float**]
- ♦ **CX** = cx, [**float**]
- ♦ **CY** = cy, [**float**]
- ♦ **CXY** = cxy, [**float**]
- ♦ **A01** = a01, [**float**]
- ♦ **A02** = a02, [**float**]
- ♦ **AMX3** = amx3, [**float**]
- ♦ **T0** = t0, [**float**]

) # *fin OB_ELO*

RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young drainé dans la direction 1.

E2 : réel donnant la valeur du module d'Young drainé dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson drainé dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson drainé dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
ROF : réel donnant la valeur de la masse volumique du fluide.
PHI : réel donnant la valeur de la porosité.
M : réel donnant la valeur du module d'incompressibilité de Biot.
PX : réel donnant la valeur du coefficient de perméabilité suivant Ox.
PY : réel donnant la valeur du coefficient de perméabilité suivant Oy .
PXY : réel donnant la valeur du coefficient de perméabilité suivant Oxy.
B1 : réel donnant la valeur du coefficient de Biot dans la direction 1.
B2 : réel donnant la valeur du coefficient de Biot dans la direction 2.
CE0 : réel donnant la valeur de la chaleur volumique à déformations constantes drainées.
CX : réel donnant la valeur du coefficient de conductivité thermique suivant Ox.
CY : réel donnant la valeur du coefficient de conductivité thermique suivant Oy.
CXY : réel donnant la valeur du coefficient de conductivité thermique suivant Oxy.
A01 : réel donnant la valeur du coefficient thermoélastique de compressibilité à déformation volumique nulle et en condition drainée du matériau dans la direction 1.
A02 : réel donnant la valeur du coefficient thermoélastique de compressibilité à déformation volumique nulle et en condition drainée du matériau dans la direction 2.
AMX3 : réel donnant la valeur du coefficient de dilatation à déformation volumique nulle en condition drainée.
T0 : réel donnant la valeur de la température de référence.

3.24.4 Classe OB_MC

La classe **OB_MC** permet d'affecter un comportement de type Mohr-Coulomb aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

mc = **OB_MC**(
- ♦ **V** = v, [**OB_ELI**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OB_MC*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.5 Classe OB_VMSE

La classe **OB_VMSE** permet d'affecter un comportement de type Von Mises sans écrouissage aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

vmse = **OB_VMSE**(
♦ **V** = v, [**OB_ELI**]
♦ **K** = k, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OB_VMSE*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
K : réel donnant la valeur de la résistance en cisaillement simple.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.6 Classe OB_VMAE

La classe **OB_VMAE** permet d'affecter un comportement de type Von Mises avec écrouissage aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

vmae = **OB_VMAE**(
♦ **V** = v, [**OB_ELI**]
♦ **K** = k, [**float**]
♦ **H** = h, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OB_VMAE*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
K : réel donnant la valeur de la résistance en cisaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.7 Classe OB_DPSE

La classe **OB_DPSE** permet d'affecter un comportement de type Drucker-Prager sans écrouissage aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

dpse = **OB_DPSE**(
♦ **V** = v, [**OB_ELI**]
♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OB_DPSE*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.8 Classe OB_DPAE

La classe **OB_DPAE** permet d'affecter un comportement de type Drucker-Prager avec écrouissage aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

dpae = **OB_DPAE**(
- ♦ **V** = v, [**OB_ELI**]
- ♦ **C** = c, [**float**]
- ♦ **PHI** = phi, [**float**]
- ♦ **PSI** = psi, [**float**]
- ♦ **XHI** = xhi, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OB_DPAE*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
XHI : réel donnant la valeur du paramètre d'écrouissage.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.9 Classe OB_CP

La classe **OB_CP** permet d'affecter un comportement de type critère parabolique aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

cp = **OB_CP**(
- ♦ **V** = v, [**OB_ELI**]
- ♦ **RC** = rc, [**float**]
- ♦ **RT** = rt, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OB_CP*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple..
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.10 Classe OB_VE

La classe **OB_VE** permet d'affecter un comportement de type Vermeer aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

ve = **OB_VE**(
- ♦ **V** = v, [**OB_ELI**]
- ♦ **EPS0** = eps0, [**float**]
- ♦ **PHICV** = phicv, [**float**]

♦ **PHIP** = phip, [**float**]
♦ **B0** = b0, [**float**]
♦ **EPSC0** = epsc0, [**float**]
♦ **P0** = p0, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OB_VE*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
EPS0 : réel donnant la valeur de déformation volumique élastique initiale.
PHICV : réel donnant la valeur de l'angle de frottement à l'état critique.
PHIP : réel donnant la valeur de l'angle de frottement au pic.
B0 : réel donnant la valeur d'un paramètre du modèle.
EPSC0 : réel donnant la valeur d'un paramètre du modèle.
P0 : réel donnant la valeur de la pression de référence.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.11 Classe OB_NO

La classe **OB_NO** permet d'affecter un comportement de type Nova aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

no = **OB_NO**(
♦ **V** = v, [**OB_ELI**]
♦ **B0** = b0, [**float**]
♦ **L0** = l0, [**float**]
♦ **M** = m, [**float**]
♦ **L** = l, [**float**]
♦ **D** = d, [**float**]
♦ **MM** = mm, [**float**]
♦ **MU** = mu, [**float**]
♦ **PCO** = pco, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OB_NO*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
B0 : réel donnant la valeur de la pente initiale de déchargement.
L0 : réel donnant la valeur de la pente initiale de contrainte-déformation.
M : réel donnant la valeur d'un paramètre du modèle.
L : réel donnant la valeur d'un paramètre du modèle.
D : réel donnant la valeur d'un paramètre du modèle.
MM : réel donnant la valeur de la pente de la droite 9/P.
MU : réel donnant la valeur d'un paramètre du modèle.
PCO : réel donnant la valeur du paramètre définissant la surface de charge initiale.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.12 Classe OB_CCM

La classe **OB_CCM** permet d'affecter un comportement de type Cam Clay modifié aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

ccm = **OB_CCM**(
- ♦ **V** = v, [**OB_ELI**]
- ♦ **ALOE** = aloe, [**float**]
- ♦ **AKOE** = akoe, [**float**]
- ♦ **AMC** = amc, [**float**]
- ♦ **OED** = oed, [**float**]
- ♦ **PCO** = pco, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OB_CCM*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
ALOE : réel donnant la valeur de la pente de la courbe de consolidation vierge.
AKOE : réel donnant la valeur de la pente des courbes charge-décharge.
AMC : réel donnant la valeur de pente de la courbe d'état critique.
OED : réel donnant la valeur de l'indice des vides initial.
PCO : réel donnant la valeur de la pression de préconsolidation initiale.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.13 Classe OB_PH

La classe **OB_PH** permet d'affecter un comportement de type Prévost et Hoeg aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

ph = **OB_PH**(
- ♦ **V** = v, [**OB_ELI**]
- ♦ **A0** = a0, [**float**]
- ♦ **B0** = b0, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OB_PH*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
A0 : réel donnant la valeur d'un paramètre du modèle.
B0 : réel donnant la valeur d'un paramètre du modèle.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.24.14 Classe OB_CO

La classe **OB_CO** permet d'affecter un comportement de type critère orienté aux éléments de thermo-mécanique bidimensionnelle des milieux poreux.

co = **OB_CO**(
- ♦ **V** = v, [**OB_ELI**]

♦ **C** = c, [**float**]
♦ **PHI** = phi, [**float**]
♦ **PSI** = psi, [**float**]
♦ **ALPHA** = alpha, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OB_CO*

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
ALPHA : réel donnant la valeur de l'angle par rapport à l'axe Ox.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25 Données relatives au modèle tridimensionnel de thermo-mécanique des milieux poreux

3.25.1 Classe OT

La classe **OT** permet de définir les caractéristiques d'un groupe d'éléments pour la thermo-mécanique tridimensionnelle des milieux poreux.

ot = **OT**(
♦ **NOMG** = nomg, [**str**]
♦ **ACTI** = / **'A'**,
♦ **IMOD** = / **1**,
♦ **CARA** = cara, [**OT_ELI**]
/ **2**,
♦ **CARA** = cara, [**OT_ELO**]
/ **11**,
♦ **CARA** = cara, [**OT_VMSE**]
/ **12**,
♦ **CARA** = cara, [**OT_VMAE**]
/ **13**,
♦ **CARA** = cara, [**OT_DPSE**]
/ **14**,
♦ **CARA** = cara, [**OT_DPAE**]
/ **15**,
♦ **CARA** = cara, [**OT_CP**]
/ **16**,
♦ **CARA** = cara, [**OT_VE**]
/ **17**,
♦ **CARA** = cara, [**OT_NO**]
/ **18**,
♦ **CARA** = cara, [**OT_CCM**]
/ **19**,
♦ **CARA** = cara, [**OT_PH**]

/ 41,
♦ CARA = cara, [OT_CO]
) # *fin OT*

NOMG : chaîne de caractères donnant le nom du groupe.
ACTI : chaîne de caractères donnant le caractère actif ('A') ou inactif ('I') du groupe.
IMOD : entier donnant le type de modèle mécanique utilisé pour les éléments du groupe à choisir parmi 1, 2, 11, 12, 13, 14, 15, 16, 17, 18, 19, 41.
CARA : objet de type à choisir parmi :
- OT_ELI pour une loi de type thermo-poro-élasticité linéaire isotrope,
- OT_ELO pour une loi de type thermo-poro-élasticité linéaire orthotrope de révolution,
- OT_VMSE pour une loi thermo-poro-élastoplastique de type Von Mises sans écrouissage,
- OT_VMAE pour une loi thermo-poro-élastoplastique de type Von Mises avec écrouissage,
- OT_DPSE pour une loi thermo-poro-élastoplastique de type Drucker-Prager sans écrouissage,
- OT_DPAE pour une loi thermo-poro-élastoplastique de type Drucker-Prager avec écrouissage,
- OT_CP pour une loi thermo-poro-élastoplastique de type critère parabolique,
- OT_VE pour une loi thermo-poro-élastoplastique de type Vermeer,
- OT_NO pour une loi thermo-poro-élastoplastique de type Nova,
- OT_CCM pour une loi thermo-poro-élastoplastique de type Cam Clay modifié,
- OT_PH pour une loi thermo-poro-élastoplastique de type Prevost-Hoeg,
- OT_CO pour une loi thermo-poro-élastoplastique de type critère orienté,

en cohérence avec le choix effectué pour IMOD.

3.25.2 Classe OT_ELI

La classe OT_ELI permet d'affecter un comportement de type thermo-poro-élasticité linéaire isotrope aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

eli = OT_ELI(
♦ RO = ro, [float]
♦ YOUNG = young, [float]
♦ POISS = poiss, [float]
♦ ROF = rof, [float]
♦ PHI = phi, [float]
♦ M = m, [float]
♦ PX = px, [float]
♦ PY = py, [float]
♦ PZ = pz, [float]
♦ PXY = pxy, [float]
♦ PYZ = pyz, [float]
♦ PXZ = pxz, [float]
♦ B = b, [float]
♦ CE0 = ce0, [float]
♦ CX = cx, [float]
♦ CY = cy, [float]

♦ **CZ** = cz, [**float**]
♦ **CXY** = cxy, [**float**]
♦ **CYZ** = cyz, [**float**]
♦ **CXZ** = cxz, [**float**]
♦ **A0** = a0, [**float**]
♦ **AMX3** = amx3, [**float**]
♦ **T0** = t0, [**float**]
) # *fin OT_ELI*

RO : réel donnant la valeur de la masse volumique.
YOUNG : réel donnant la valeur du module d'Young drainé.
POISS : réel donnant la valeur du coefficient de Poisson drainé.
ROF : réel donnant la valeur de la masse volumique du fluide.
PHI : réel donnant la valeur de la porosité.
M : réel donnant la valeur du module d'incompressibilité de Biot.
PX : réel donnant la valeur du coefficient de perméabilité suivant Ox.
PY : réel donnant la valeur du coefficient de perméabilité suivant Oy .
PZ : réel donnant la valeur du coefficient de perméabilité suivant Oz .
PXY : réel donnant la valeur du coefficient de perméabilité suivant Oxy.
PYZ : réel donnant la valeur du coefficient de perméabilité suivant Oyz.
PXZ : réel donnant la valeur du coefficient de perméabilité suivant Oxz.
B : réel donnant la valeur du coefficient de Biot.
CE0 : réel donnant la valeur de la chaleur volumique à déformations constantes drainées.
CX : réel donnant la valeur du coefficient de conductivité thermique suivant Ox.
CY : réel donnant la valeur du coefficient de conductivité thermique suivant Oy.
CZ : réel donnant la valeur du coefficient de conductivité thermique suivant Oy.
CXY : réel donnant la valeur du coefficient de conductivité thermique suivant Oxy.
CYZ : réel donnant la valeur du coefficient de conductivité thermique suivant Oyz.
CXZ : réel donnant la valeur du coefficient de conductivité thermique suivant Oxz.
A0 : réel donnant la valeur du coefficient thermoélastique de compressibilité à déformation volumique nulle et en condition drainée du matériau.
AMX3 : réel donnant la valeur du coefficient de dilatation à déformation volumique nulle en condition drainée.
T0 : réel donnant la valeur de la température de référence.

3.25.3 Classe OT_ELO

La classe **OT_ELO** permet d'affecter un comportement de type thermo-poro-élasticité linéaire orthotrope aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

elo = **OT_ELO**(
♦ **RO** = ro, [**float**]
♦ **E1** = e1, [**float**]
♦ **E2** = e2, [**float**]
♦ **P1** = p1, [**float**]
♦ **P2** = p2, [**float**]
♦ **G2** = g2, [**float**]
♦ **TETA** = teta, [**float**]
♦ **PSI** = psi, [**float**]

♦ **ROF** = rof, [**float**]
♦ **PHI** = phi, [**float**]
♦ **M** = m, [**float**]
♦ **PX** = px, [**float**]
♦ **PY** = py, [**float**]
♦ **PZ** = pz, [**float**]
♦ **PXY** = pxy, [**float**]
♦ **PYZ** = pyz, [**float**]
♦ **PXZ** = pxz, [**float**]
♦ **B1** = b1, [**float**]
♦ **B2** = b2, [**float**]
♦ **CE0** = ce0, [**float**]
♦ **CX** = cx, [**float**]
♦ **CY** = cy, [**float**]
♦ **CZ** = cz, [**float**]
♦ **CXY** = cxy, [**float**]
♦ **CYZ** = cyz, [**float**]
♦ **CXZ** = cxz, [**float**]
♦ **A01** = a01, [**float**]
♦ **A02** = a02, [**float**]
♦ **AMX3** = amx3, [**float**]
♦ **T0** = t0, [**float**]
) # *fin OT_ELO*

RO : réel donnant la valeur de la masse volumique.
E1 : réel donnant la valeur du module d'Young drainé dans la direction 1.
E2 : réel donnant la valeur du module d'Young drainé dans la direction 2.
P1 : réel donnant la valeur du coefficient de Poisson drainé dans la direction 1.
P2 : réel donnant la valeur du coefficient de Poisson drainé dans la direction 2.
G2 : réel donnant la valeur du module de cisaillement.
TETA : réel donnant la valeur de l'angle entre l'axe Ox et la direction 1.
ROF : réel donnant la valeur de la masse volumique du fluide.
PHI : réel donnant la valeur de la porosité.
M : réel donnant la valeur du module d'incompressibilité de Biot.
PX : réel donnant la valeur du coefficient de perméabilité suivant Ox.
PY : réel donnant la valeur du coefficient de perméabilité suivant Oy .
PZ : réel donnant la valeur du coefficient de perméabilité suivant Oz .
PXY : réel donnant la valeur du coefficient de perméabilité suivant Oxy.
PYZ : réel donnant la valeur du coefficient de perméabilité suivant Oyz.
PXZ : réel donnant la valeur du coefficient de perméabilité suivant Oxz.
B1 : réel donnant la valeur du coefficient de Biot dans la direction 1.
B2 : réel donnant la valeur du coefficient de Biot dans la direction 2.
CE0 : réel donnant la valeur de la chaleur volumique à déformations constantes drainées.
CX : réel donnant la valeur du coefficient de conductivité thermique suivant Ox.
CY : réel donnant la valeur du coefficient de conductivité thermique suivant Oy.
CZ : réel donnant la valeur du coefficient de conductivité thermique suivant Oz.
CXY : réel donnant la valeur du coefficient de conductivité thermique suivant Oxy.
CYZ : réel donnant la valeur du coefficient de conductivité thermique suivant Oyz.
CXZ : réel donnant la valeur du coefficient de conductivité thermique suivant Oxz.

A01 : réel donnant la valeur du coefficient thermoélastique de compressibilité à déformation volumique nulle et en condition drainée du matériau dans la direction 1.
A02 : réel donnant la valeur du coefficient thermoélastique de compressibilité à déformation volumique nulle et en condition drainée du matériau dans la direction 2.
AMX3 : réel donnant la valeur du coefficient de dilatation à déformation volumique nulle en condition drainée.
T0 : réel donnant la valeur de la température de référence.

3.25.4 Classe OT_VMSE

La classe **OT_VMSE** permet d'affecter un comportement de type Von Mises sans écrouissage aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

vmse = **OT_VMSE**(
♦ **V** = v, [**OT_ELI**]
♦ **K** = k, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OT_VMSE*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
K : réel donnant la valeur de la résistance en cisaillement simple.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.5 Classe OT_VMAE

La classe **OT_VMAE** permet d'affecter un comportement de type Von Mises avec écrouissage aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

vmae = **OT_VMAE**(
♦ **V** = v, [**OT_ELI**]
♦ **K** = k, [**float**]
♦ **H** = h, [**float**]
♦ **BETA** = beta, [**float**]
) # *fin OT_VMAE*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
K : réel donnant la valeur de la résistance en cisaillement simple.
H : réel donnant la valeur de la pente de la droite d'essai uniaxial.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.6 Classe OT_DPSE

La classe **OT_DPSE** permet d'affecter un comportement de type Drucker-Prager sans écrouissage aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

dpse = **OT_DPSE**(
- **V** = v, [**OT_ELI**]
- **C** = c, [**float**]
- **PHI** = phi, [**float**]
- **PSI** = psi, [**float**]
- **BETA** = beta, [**float**]

) # *fin OT_DPSE*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.7 Classe OT_DPAE

La classe **OT_DPAE** permet d'affecter un comportement de type Drucker-Prager avec écrouissage aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

dpae = **OT_DPAE**(
- **V** = v, [**OT_ELI**]
- **C** = c, [**float**]
- **PHI** = phi, [**float**]
- **PSI** = psi, [**float**]
- **XHI** = xhi, [**float**]
- **BETA** = beta, [**float**]

) # *fin OT_DPAE*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
XHI : réel donnant la valeur du paramètre d'écrouissage.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.8 Classe OT_CP

La classe **OT_CP** permet d'affecter un comportement de type critère parabolique aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

cp = **OT_CP**(
- **V** = v, [**OT_ELI**]
- **RC** = rc, [**float**]
- **RT** = rt, [**float**]
- **BETA** = beta, [**float**]

) # *fin OT_CP*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
RC : réel donnant la valeur de la résistance en compression simple.
RT : réel donnant la valeur de la résistance en traction simple..
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.9 Classe OT_VE

La classe **OT_VE** permet d'affecter un comportement de type Vermeer aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

ve = **OT_VE**(
- **V** = v, [**OT_ELI**]
- **EPS0** = eps0, [**float**]
- **PHICV** = phicv, [**float**]
- **PHIP** = phip, [**float**]
- **B0** = b0, [**float**]
- **EPSC0** = epsc0, [**float**]
- **P0** = p0, [**float**]
- **BETA** = beta, [**float**]

) # *fin OT_VE*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
EPS0 : réel donnant la valeur de déformation volumique élastique initiale.
PHICV : réel donnant la valeur de l'angle de frottement à l'état critique.
PHIP : réel donnant la valeur de l'angle de frottement au pic.
B0 : réel donnant la valeur d'un paramètre du modèle.
EPSC0 : réel donnant la valeur d'un paramètre du modèle.
P0 : réel donnant la valeur de la pression de référence.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.10 Classe OT_NO

La classe **OT_NO** permet d'affecter un comportement de type Nova aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

no = **OT_NO**(
- **V** = v, [**OT_ELI**]
- **B0** = b0, [**float**]
- **L0** = l0, [**float**]
- **M** = m, [**float**]
- **L** = l, [**float**]
- **D** = d, [**float**]
- **MM** = mm, [**float**]
- **MU** = mu, [**float**]
- **PCO** = pco, [**float**]
- **BETA** = beta, [**float**]

) # *fin OT_NO*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
B0 : réel donnant la valeur de la pente initiale de déchargement.
L0 : réel donnant la valeur de la pente initiale de contrainte-déformation.
M : réel donnant la valeur d'un paramètre du modèle.
L : réel donnant la valeur d'un paramètre du modèle.
D : réel donnant la valeur d'un paramètre du modèle.
MM : réel donnant la valeur de la pente de la droite 9/P.
MU : réel donnant la valeur d'un paramètre du modèle.
PCO : réel donnant la valeur du paramètre définissant la surface de charge initiale.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.11 Classe OT_CCM

La classe **OT_CCM** permet d'affecter un comportement de type Cam Clay modifié aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

ccm = **OT_CCM**(
- ♦ **V** = v, [**OT_ELI**]
- ♦ **ALOE** = aloe, [**float**]
- ♦ **AKOE** = akoe, [**float**]
- ♦ **AMC** = amc, [**float**]
- ♦ **OED** = oed, [**float**]
- ♦ **PCO** = pco, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OT_CCM*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
ALOE : réel donnant la valeur de la pente de la courbe de consolidation vierge.
AKOE : réel donnant la valeur de la pente des courbes charge-décharge.
AMC : réel donnant la valeur de pente de la courbe d'état critique.
OED : réel donnant la valeur de l'indice des vides initial.
PCO : réel donnant la valeur de la pression de préconsolidation initiale.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.12 Classe OT_PH

La classe **OT_PH** permet d'affecter un comportement de type Prévost et Hoeg aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

ph = **OT_PH**(
- ♦ **V** = v, [**OT_ELI**]
- ♦ **A0** = a0, [**float**]
- ♦ **B0** = b0, [**float**]
- ♦ **BETA** = beta, [**float**]

) # *fin OT_PH*

V : objet de type **OT_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.

A0 : réel donnant la valeur d'un paramètre du modèle.
B0 : réel donnant la valeur d'un paramètre du modèle.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.25.13 Classe OT_CO

La classe **OT_CO** permet d'affecter un comportement de type critère orienté aux éléments de thermo-mécanique tridimensionnelle des milieux poreux.

```
co = OT_CO(
    ♦ V = v, [OT_ELI]
    ♦ C = c, [float]
    ♦ PHI = phi, [float]
    ♦ PSI = psi, [float]
    ♦ U = u, [list<float>]
    ♦ BETA = beta, [float]
    ) # fin OT_CO
```

V : objet de type **OB_ELI** définissant les caractéristiques thermo-poro-élastiques isotropes.
C : réel donnant la valeur de la cohésion.
PHI : réel donnant la valeur de l'angle de frottement interne.
PSI : réel donnant la valeur de l'angle de dilatance.
U : liste de 3 réels donnant les coordonnées dans le repère du maillage du vecteur normal au plan de discontinuité.
BETA : réel donnant la valeur du coefficient du tenseur des contraintes effectives plastiques.

3.26 Données relatives aux conditions limites

3.26.1 Classe COND

La classe **COND** permet de définir des conditions aux limites sur l'inconnue principale.

```
cond = COND(
    ♦ M = / 0,
          / 1,
          / 2,
    ♦ OPT = opt, [list <| COND_DDL
                       | COND_IMP
                       | COND_NUL
                       | COND_REP
                    >] # fin list
    ) # fin COND
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
OPT : liste d'objets de type à choisir parmi :
· **COND_DDL** pour imposer des ddl par numéro d'équation,

· **COND_IMP** pour imposer des ddl non nuls par noeud,
· **COND_NUL** pour imposer des ddl nuls par noeud,
· **COND_REP** pour imposer des changements de repère,
permettant d'introduire les conditions limites par option.

3.26.2 Classe COND_DDL

La classe **COND_DDL** permet de définir des degrés de liberté imposés par numéro d'équation.

```
cond_ddl = COND_DDL(
           ♦ NCLT = nclt, [int]
           ♦ NCLNN = nclnn, [int]
           ♦ NCR = ncr, [int]
           ◊ # si NCLT ≠ 0
             ♦ NEQ = neq, [list<int>]
             ♦ UIMP = uimp, [list<int>]
           ◊ # si NCR ≠ 0
             ♦ KCR = kcr, [list<int>]
             ♦ K = / 1,
                   / 2,
             ♦ ANG = ang, [list<float>]
           ) # fin COND_DDL
```

NCLT : entier donnant le nombre total de conditions limites.
NCLNN : entier donnant le nombre de conditions limites non nulles.
NCR : entier donnant le nombre de changements de repères.
NEQ : liste d'entiers donnant les numéros d'équation des **NCLT** ddl dont on impose la valeur.
UIMP : liste de réels donnant les valeurs des **NCLT** ddl imposés.
KCR : liste d'entiers donnant les numéros des **NCR** noeuds pour lesquels on effectue un changement de repère.
K : entier indiquant le caractère bidimensionnel (**1**) ou tridimensionnel (**2**) du problème
ANG : liste de réels de taille **NCR.K** donnant les angles définissant les changements de repère.

3.26.3 Classe COND_IMP

La classe **COND_IMP** permet de définir des degrés de liberté dont la valeur imposée est non nulle par noeuds.

```
cond_imp = COND_IMP(
           ♦ GEN = gen, [list<GEN_IMP>]
           ) # fin COND_IMP
```

GEN : liste d'objets de type **GEN_IMP** permettant d'imposer des conditions limites non nulles.

Génération de ddl imposés non nuls. La classe **GEN_IMP** permet de générer un ensemble de degrés de liberté imposés.

gen_imp = **GEN_IMP**(
♦ **IGEN** = / **1**,
♦ **ID** = id, [**int**]
♦ **IF** = if, [**int**]
♦ **IPAS** = ipas, [**int**]
/ **2**,
♦ **NP** = np, [**int**]
♦ **NUM** = num, [**list**<**int**>]
♦ **IL** = il, [**int**]
♦ **UIMP** = uimp, [**list**<**float**>]
) # *fin GEN_IMP*

IGEN : entier indiquant le mode de génération de l'ensemble des noeuds :
— **1** : pour une génération par **ID**, **IF** et **IPAS**,
— **2** : pour une génération par **NP** et **NUM**.
ID : entier donnant le premier numéro à générer.
IF : entier donnant le dernier numéro à générer.
IPAS : entier donnant le pas de progression entre **ID** et **IF**.
NP : entier donnant le nombre de numéros de noeuds à définir.
NUM : liste d'entiers donnant les **NP** numéros des noeuds.
IL : entier donnant le numéro du ddl imposé.
UIMP : liste de réels donnant les **NP** valeurs (non nulles) des ddl imposés

3.26.4 Classe COND_NUL

La classe **COND_NUL** permet de définir des degrés de liberté dont la valeur imposée est nulle par noeuds.

cond_nul = **COND_NUL**(
♦ **GEN** = gen, [**list**<**GEN_NUL**>]
) # *fin COND_NUL*

GEN : liste d'objets de type **GEN_NUL** permettant d'imposer des conditions limites nulles.

Génération de ddl imposés nuls. La classe **GEN_NUL** permet de générer un ensemble de degrés de liberté imposés nuls.

gen_nul = **GEN_NUL**(
♦ **IGEN** = / **1**,
♦ **ID** = id, [**int**]
♦ **IF** = if, [**int**]
♦ **IPAS** = ipas, [**int**]
/ **2**,
♦ **NP** = np, [**int**]
♦ **NUM** = num, [**list**<**int**>]

♦ **IDL** = idl, [**list** < | **0**
| **1**
>] # *fin list*
♦ **NDLN** = ndln, [**int**]
) # *fin GEN_NUL*

IGEN : entier indiquant le mode de génération de l'ensemble des noeuds :
— **1** : pour une génération par **ID**, **IF** et **IPAS**,
— **2** : pour une génération par **NP** et **NUM**.
ID : entier donnant le premier numéro à générer.
IF : entier donnant le dernier numéro à générer.
IPAS : entier donnant le pas de progression entre **ID** et **IF**.
NP : entier donnant le nombre de numéros de noeuds à définir.
NUM : liste d'entiers donnant les **NP** numéros des noeuds.
IDL : liste d'entiers de taille **NDLN** permettant de définir le caractère imposé (**1**) ou pas (**0**) des ddl.
NDLN : entier donnant le nombre maximum de ddl par noeud.

3.26.5 Classe COND_REP

La classe **COND_REP** permet de définir des changements de repère.

cond_rep = **COND_REP**(
♦ **NCR** = ncr, [**int**]
♦ **KCR** = kcr, [**list**<**int**>]
◊ # *si problème bidimensionnel*
♦ **NR** = nr, [**int**]
♦ **A** = a, [**list**<**float**>]
◊ # *si problème tridimensionnel*
♦ **NR** = nr, [**int**]
♦ **A** = a, [**list**<**float**>]
♦ **B** = b, [**list**<**float**>]
♦ **JCR** = jcr, [**list**<**int**>]
) # *fin COND_REP*

NCR : entier donnant le nombre de changements de repère.
KCR : liste d'entiers donnant les **NCR** numéros des noeuds affectés d'un changement de repère.
NR : entier donnant le nombre d'angles différents définissant les changements de repère.
A : liste de réels donnant **NR** angles intervenant dans la définition des changements de repère.
B : liste de réels donnant **NR** angles intervenant dans la définition des changements de repère.
JCR : liste d'entiers donnant les numéros de l'angle (rang dans **A**) ou du couple d'angles (rang dans **A** et **B**) définissant les **NCR** changements de repère aux noeuds.

3.27 Données relatives aux chargements

3.27.1 Classe CHAR

La classe **CHAR** permet de définir un cas de chargement.

```
char = CHAR(
        ♦ M = / 0,
              / 1,
              / 2,
              / 3,
              / 4,
        ♦ OPT = opt, [list <| CHAR_CNU
                           | CHAR_CUR
                           | CHAR_DVU
                           | CHAR_ECH
                           | CHAR_EFD
                           | CHAR_FNU
                           | CHAR_FOS
                           | CHAR_FUR
                           | CHAR_LAM
                           | CHAR_PHS
                           | CHAR_PNC
                           | CHAR_PNU
                           | CHAR_POI
                           | CHAR_SOL
                           | CHAR_PUR
                           | CHAR_SIG
                        >] # fin list
        ) # fin CHAR
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**, **3**, **4**.
OPT : liste d'objets de type à choisir parmi :

- **CHAR_CNU** pour un chargement de type cisaillement non uniforme,
- **CHAR_CUR** pour un chargement de type cisaillement uniformement réparti,
- **CHAR_DVU** pour un chargement de type débit volumique uniforme,
- **CHAR_ECH** pour un chargement de type flux associé à une condition d'échange linéaire,
- **CHAR_EFD** pour un chargement de type chargement d'effet différé,
- **CHAR_FNU** pour un chargement de type flux non uniforme,
- **CHAR_FOS** pour un chargement de type forces de surface
- **CHAR_FUR** pour un chargement de type flux uniformement réparti,
- **CHAR_LAM** pour un chargement de type forces de déconfinement,
- **CHAR_PHS** pour un chargement de type pression hydrostatique,
- **CHAR_PNC** pour un chargement de type pression non coincidente,
- **CHAR_PNU** pour un chargement de type pression non uniforme,
- **CHAR_POI** pour un chargement de type poids volumique,
- **CHAR_PUR** pour un chargement de type pression uniformement répartie,
- **CHAR_SIG** pour un chargement de type état de contraintes initiales,

· **CHAR_SOL** pour un chargement de type sollicitations nodales,
permettant d'introduire les chargements par option.

3.27.2 Classe CHAR_CNU

La classe **CHAR_CNU** permet de définir un chargement de type cisaillement non uniforme.

cnu = **CHAR_CNU**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **CSVAR** = csvar, [**list**<**float**>]
) # *fin CHAR_CNU*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
CSVAR : liste de réels de taille **NF**.**NMF** donnant les valeurs du cisaillement aux noeuds des facettes.

3.27.3 Classe CHAR_CUR

La classe **CHAR_CUR** permet de définir un chargement de type cisaillement uniformément réparti.

cur = **CHAR_CUR**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **CS** = cs, [**float**]
) # *fin CHAR_CUR*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
CS : réel donnant la valeur du cisaillement.

3.27.4 Classe CHAR_DVU

La classe **CHAR_DVU** permet de définir un chargement de type débit volumique uniforme.

dvu = **CHAR_DVU**(
♦ **NG** = ng, [**int**]
♦ **KG** = kg, [**list**<**int**>]
♦ **DV** = dv, [**float**]
) # *fin CHAR_DVU*

NG : entier donnant le nombre de groupes d'élément à débits volumiques imposés.

KG : liste d'entiers donnant les numéros de groupes d'élément à débits volumiques imposés.
DV : réel donnant le débit volumique imposé.

3.27.5 Classe CHAR_ECH

La classe **CHAR_ECH** permet de définir un chargement de flux associé à une condition d'échange linéaire.

ech = **CHAR_ECH**(
♦ **NG** = ng, [**int**]
♦ **KG** = kg, [**list**<**int**>]
♦ **TETAE** = tetae, [**list**<**float**>]
) # *fin CHAR_ECH*

NG : entier donnant le nombre de groupes d'élément d'échange à comportement linéaire.
KG : liste d'entiers donnant les numéros de groupes d'élément d'échange à comportement linéaire.
TETAE : liste de réels donnant les valeurs de la temperature extérieure en bordure des groupes d'élément d'échange à comportement linéaire.

3.27.6 Classe CHAR_EFD

La classe **CHAR_EFD** permet de définir un chargement d'effet différé dans les éléments de massif.

efd = **CHAR_EFD**(
♦ **M1** = / **0**,
♦ **NC** = nc, [**int**]
◊ **DEFC** = defc, [**list**<**EFD_COUC**>]
♦ **NGA** = nga, [**int**]
◊ **KGA** = kga, [**list**<**int**>]
/ **1**,
♦ **NOMFI** = nomfi, [**str**]
♦ **NOMFA** = nomfa, [**str**]
♦ **NG** = ng, [**int**]
◊ **CARAI** = carai, [**list**<**EFD_INST**>]
) # *fin CHAR_EFD*

M1 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier pour la lecture de l'état de contraintes initial.
NC : entier donnant le nombre de couches du terrain intervenant pour l'état de contraintes inital.
DEFC : liste d'objets de type **EFD_COUC** permettant de définir les **NC** couches.
NGA : entier donnant le nombre de groupes où on annule les contraintes géostatiques.
KGA : liste d'entiers donnant les numéro des **NGA** groupes où on annule les contraintes géostatiques.
NOMFI : chaîne de caractère donnant le nom du fichier de résultats CESAR donnant l'état de contraintes initial.

NOMFA : chaîne de caractère donnant le nom du fichier de résultats CESAR donnant l'état de contraintes actuel.
NG : entier donnant le nombre de groupes avec effet différé.
CARAI : liste d'objets de type **EFD_INST** donnant les caractéristiques instantanées

Couches de terrain. La classe **EFD_COUC** permet de décrire chaque couche de terrain.

couc = **EFD_COUC**(
- **HSUP** = hsup, [**float**]
- **PV** = pv, [**float**]
- **CPL1** = cpl1, [**float**]
- **CPL2** = cpl2, [**float**]

) # *fin EFD_COUC*

HSUP : réel donnant la cote de la limie supérieure de la couche.
PV : réel donnant le poids volumique de la couche.
CPL1 : réel donnant le coefficient de poussée latérale selon Ox.
CPL2 : réel donnant le coefficient de poussée latérale selon Oz (déformation plane) ou Oy (tridimensionnel).

Caractéristiques instantanées. La classe **EFD_INST** permet d'affecter des caractéristiques matériaux instantanées à un groupe d'éléments.

inst = **EFD_INST**(
- **IG** = ng, [**int**]
- **YOUNGI** = youngi, [**float**]
- **POISSI** = poissi, [**float**]

) # *fin EFD_INST*

IG : entier donnant le numéro du groupe considéré.
YOUNGI : réel donnnant la valeur instantanée du module d'Young.
POISSI : réel donnnant la valeur instantanée du coefficient de Poisson.

3.27.7 Classe CHAR_FNU

La classe **CHAR_FNU** permet de définir un chargement de type flux non uniforme.

fnu = **CHAR_FNU**(
- **NF** = nf, [**int**]
- **NMF** = nmf, [**int**]
- **NFP** = nfp, [**list**<**int**>]
- **FL** = fl, [**list**<**float**>]

) # *fin CHAR_FNU*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
FL : liste de réels de taille **NF**.**NMF** donnant les valeurs du flux aux noeuds des facettes.

3.27.8 Classe CHAR_FOS

La classe **CHAR_FOS** permet de définir un chargement de type forces de surface quelconques.

fos = **CHAR_FOS**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **NDIM** = / **2**,
/ **3**,
♦ **FS** = fs, [**list**<**float**>]
) # *fin CHAR_FOS*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
NDIM : entier indiquant le caractère bidimensionnel (**2**) ou tridimensionnel (**3**) du problème.
FS : liste de réels de taille **NF**.**NMF**.**NDIM** donnant les valeurs des **NDIM** composantes des forces de surface aux noeuds des facettes.

3.27.9 Classe CHAR_FUR

La classe **CHAR_FUR** permet de définir un chargement de type flux uniformément réparti.

fur = **CHAR_FUR**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **FL** = fl, [**float**]
) # *fin CHAR_FUR*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
FL : réel donnant la valeur du flux.

3.27.10 Classe CHAR_LAM

La classe **CHAR_LAM** permet de définir un chargement de type forces de déconfinement pour les éléments de massif.

lam = **CHAR_LAM**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **NG** = ng, [**int**]
♦ **KG** = kg, [**list**<**int**>]
♦ **M1** = / **0**,

♦ **NC** = nc, [**int**]
◊ **DEFC** = defc, [**list**<**LAM_COUC**>]
♦ **NGA** = nga, [**int**]
◊ **KGA** = kga, [**list**<**int**>]
/ **1**,
♦ **NOMF** = nomf, [**str**]
◊ **DEFFD** = deffd, [**LAM_FD**]
) # *fin CHAR_LAM*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
NG : entier donnant le nombre de groupe intervenant dans le calcul des forces de surface.
KG : liste d'entiers donnant les numéros signés des groupes intervenant dans le calcul des forces de surface.
M1 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier pour la lecture de des contraintes servant à générer les forces de déconfinement.
NC : entier donnant le nombre de couches de sol.
DEFC : liste d'objets de type **LAM_COUC** permettant de définir les couches de sol.
NGA : entier donnant le nombre de groupes où on annule les contraintes d'origine géostatique.
KGA : liste d'entiers donnant les numéro des **NGA** groupes où on annule les contraintes géostatiques.
NOMF : chaîne de caractères donnant le nom du ficher de résultats CESAR contenant les contraintes servant à générer les forces de déconfinement.
DEFFD : objets de type **LAM_FD** permettant de définir le champ de pression dû à la présence d'un fluide.

Couches de terrain. La classe **LAM_COUC** permet de décrire chaque couche de terrain.

couc = **LAM_COUC**(
♦ **HSUP** = hsup, [**float**]
♦ **PV** = pv, [**float**]
♦ **CPL1** = cpl1, [**float**]
♦ **CPL2** = cpl2, [**float**]
) # *fin LAM_COUC*

HSUP : réel donnant la cote de la limie supérieure de la couche.
PV : réel donnant le poids volumique de la couche.
CPL1 : réel donnant le coefficient de poussée latérale selon Ox.
CPL2 : réel donnant le coefficient de poussée latérale selon Oz (déformation plane) ou Oy (tridimensionnel).

Pression due à un fluide. La classe **LAM_FD** permet de définir un champ de pression dû à la présence d'un fluide.

fd = **LAM_FD**(
♦ **VERTI** = verti, [**list**<**float**>]

♦ **PW** = pw, [**float**]
♦ **MFD** = / **0**,
♦ **VH** = vh, [**list**<**float**>]
/ **1**,
♦ **NOMF** = nomf, [**str**]
/ **2**,
♦ **H0** = h0, [**float**]
) # *fin LAM_FD*

VERTI : liste de 3 réels donnant les composantes du vecteur unitaire orienté suivant la verticale ascendante.
PW : réel donnant le poids volumique du fluide.
MFD : entier indiquant le mode de donnée du champ de charge hydraulique à choisir parmi :
— **0** : si on le définit dans le jeu de données,
— **1** : si on le lit dans un fichier de résultats CESAR,
— **2** : si la charge est uniforme.
VH : liste de réels donnant les valeurs de la charge hydraulique aux noeuds.
NOMF : chaîne de caractères donnant le nom du fichier de résultats CESAR.
H0 : réel donnant la valeur de la charge hydraulique.

3.27.11 Classe CHAR_DEC

La classe **CHAR_DEC** permet de définir un chargement de type forces de déconfinement.

dec = **CHAR_DEC**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
◊ # *si NF.NMF > 0*
♦ **NFP** = nfp, [**list**<**int**>]
♦ **NG** = ng, [**int**]
♦ **KG** = kg, [**list**<**int**>]
♦ **M1** = / **0**,
/ **1**,
♦ **PV** = pv, [**list**<**float**>]
◊ **DEFVA** = defva, [**DEC_VA**]
♦ **NOMF** = nomf, [**str**]
) # *fin CHAR_DEC*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
Variante 1 : on peut aussi donner pour **NF** et **NMF** deux entiers dont le produit donne le nombre de noeuds à lire dans **NFP**
Variante 2 : on peut donner **NF** et **NMF** nuls et ne pas renseigner **NFP** : le solveur constitue lui-même la liste des noeuds chargés (noeuds sur le bord des groupes dont on donne la liste via les données **NG** et **KG**).
NG : entier donnant le nombre de groupe intervenant dans le calcul des forces de surface.

KG : liste d'entiers donnant les numéros signés des groupes intervenant dans le calcul des forces de surface : signe + (resp. -) si le groupe fait partie des matériaux restants (resp. excavés).
M1 : entier indiquant la prise en compte (**1**) ou non (**0**) des forces de pesanteur dans le calcul du chargement.
DEFVA : objet de type **DEC_VA** permettant de redéfinir la densité massique des forces à distances.
PV : liste de réels donnant les poids volumique à prendre en compte pour chacun des **NG** groupes.
NOMF : chaîne de caractères donnant le nom du ficher de stockage CESAR issu d'un calcul précédent et servant au calcul du chargement.

Vecteur des forces massiques. La classe **DEC_VA** permet de définir le vecteur des forces massiques.

va = **DEC_VA**(
♦ **G** = g, [**list**<**float**>]
) # *fin DEC_VA*

G : liste de 3 réels donnant les 3 composantes dans les axes du repère de coordonnées globales du vecteur des forces massiques.

3.27.12 Classe CHAR_PHS

La classe **CHAR_PHS** permet de définir un chargement de type pression hydrostatique.

phs = **CHAR_PHS**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **H** = h, [**float**]
♦ **PVF** = pvf, [**float**]
♦ **V** = v, [**list**<**float**>]
) # *fin CHAR_PHS*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
H : réel donnant la cote de la surface libre du fluide.
PVF : réel donnant le poids volumique du fluide.
V : liste de 3 réels donnant les composantes du vecteur unitaire orienté suivant la verticale ascendante.

3.27.13 Classe CHAR_PNU

La classe **CHAR_PNU** permet de définir un chargement de type pression non uniforme.

pnu = **CHAR_PNU**(
- **NF** = nf, [**int**]
- **NMF** = nmf, [**int**]
- **NFP** = nfp, [**list**<**int**>]
- **PVAR** = pvar, [**list**<**float**>]

) # *fin CHAR_PNU*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
PVAR : liste de réels de taille **NF**.**NMF** donnant les valeurs de la pression aux noeuds des facettes.

3.27.14 Classe CHAR_PNC

La classe **CHAR_PNC** permet de définir un chargement de type pression non coincidente.

pnc = **CHAR_PNC**(
- **P** = p, [**float**]
- **ZP** = zp, [**float**]
- **LONG1** = long1, [**float**]
- **LONG2** = long2, [**float**]
- **NC** = nc, [**int**]
- **NT** = nt, [**int**]
- **DEFT** = deft, [**list**<**PNC_PAS**>]

) # *fin CHAR_PNC*

P : réel donnant la valeur de la pression uniforme appliquée.
ZP : réel donnant la cote du plan de chargement.
LONG1 : réel donnant la première longueur de chargement.
LONG2 : réel donnant la seconde longueur de chargement.
NT : entier donnant le nombre de pas de temps.
NC : entier donnant le nombre de charges à chaque pas de temps.
DEFT : liste d'objets de type **PNC_PAS** permettant de définir le paramétrage des différentes charges appliquées.

Pas de temps. La classe **PNC_PAS** permet de définir le paramétrage des **NC** charges de pression non coincidente appliquées pour un pas de temps.

pas = **PNC_PAS**(
- **XC** = xc, [**list**<**float**>]
- **YC** = yc, [**list**<**float**>]
- **THETA** = theta, [**list**<**float**>]

) # *fin PNC_PAS*

XC : liste de réels de taille **NC** donnant les abscisses des **NC** charges.
YC : liste de réels de taille **NC** donnant les ordonnées des **NC** charges.
THETA : liste de réels de taille **NC** donnant les angles des **NC** charges.

3.27.15 Classe CHAR_POI

La classe **CHAR_POI** permet de définir un chargement de type poids volumique.

poi = **CHAR_POI**(
◊ # *si on redéfinit le vecteur accélération*
♦ **NDIRFV** = ndirfv, [**int**]
♦ **FV** = fv, [**list**<**POI_FV**>]
) # *fin CHAR_POI*

NDIRFV : entier donnant le nombre de directions pour lesquelles on redéfinit l'accélération.
FV : liste d'objets de type **POI_FV** permettant de redéfinir l'accélération.

Direction d'accélération. La classe **POI_FV** permet de définir une direction d'accélération pour un groupe d'éléments.

fv = **POI_FV**(
♦ **NG** = ng, [**int**]
♦ **NUMG** = numg, [**list**<**int**>]
♦ **NDIM** = / **2**,
/ **3**,
♦ **V** = v, [**list**<**float**>]
) # *fin POI_FV*

NG : entier donnant le nombre de groupes d'éléments concernés.
NUMG : liste d'entiers donnant les numéros des groupes d'éléments concernés.
NDIM : entier indiquant le caractère bidimensionnel (**2**) ou tridimensionnel (**3**) du problème.
V : liste de **NDIM** réels donnant les composantes du vecteur accélération.

3.27.16 Classe CHAR_PUR

La classe **CHAR_PUR** permet de définir un chargement de type pression uniformément répartie.

pur = **CHAR_PUR**(
♦ **NF** = nf, [**int**]
♦ **NMF** = nmf, [**int**]
♦ **NFP** = nfp, [**list**<**int**>]
♦ **P** = p, [**float**]
) # *fin CHAR_PUR*

NF : entier donnant le nombre de facettes.
NMF : entier donnant le nombre maximum de noeuds par facettes.
NFP : liste d'entiers de taille **NF**.**NMF** donnant les numéros des noeuds des facettes.
P : réel donnant la valeur de la pression.

3.27.17 Classe CHAR_SIG

La classe **CHAR_SIG** permet de prendre en compte un état de contraintes initiales.

```
sig = CHAR_SIG(
      ♦ M1 = / 0 ,
               ♦ ICAL = / 0,
                        / 1,
                        / 2,
               ♦ IOPT = / 1,
                          ♦ OPT = opt, [SIG_CST]
                        / 2,
                          ♦ OPT = opt, [SIG_THER]
                        / 3,
                          ♦ OPT = opt, [SIG_GEO]
                        / 4,
                          ♦ OPT = opt, [SIG_ISO]
             / 1,
               ♦ NOMF1 = nomf1, [str]
      ) # fin CHAR_SIG
```

M1 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier de lecture des contraintes initiales.
ICAL : entier indiquant le mode d'utilisation des contraintes initiales à choisir parmi :
— **0** : pour initialisation des contraintes et non prise en compte du chargement,
— **1** : pour initialisation des contraintes et prise en compte du chargement,
— **2** : pour non initialisation des contraintes et prise en compte du chargement.
IOPT : entier indiquant le procédé de génération des contraintes initiales à choisir parmi :
— **1** : pour des contraintes constantes,
— **2** : pour des contraintes correspondant à un chargement thermique,
— **3** : pour des contraintes d'origine géostatique,
— **4** : pour des contraintes dues à la présence d'un fluide.
OPT : objet de type à choisir parmi :
· **SIG_CST** pour des contraintes initiales constantes,
· **SIG_THER** pour des contraintes initiales de type thermique,
· **SIG_GEO** pour des contraintes initiales de type géostatique,
· **SIG_ISO** pour des contraintes initiales de type pression isotrope,
en cohérence avec le choix effectué pour **IOPT**.

Contraintes constantes. La classe **SIG_CST** permet de générer des contraintes initiales constantes.

```
cst = SIG_CST(
      ♦ NG = ng, [int]
      ♦ IGR = igr, [list<int>]
      ♦ SIGMA = sigma, [list<float>]
      ) # fin SIG_CST
```

NG : entier donnant le nombre de groupes à contraintes initiales constantes.

IGR : liste d'entiers donnant les numéros des groupes à contraintes initiales constantes.
SIGMA : liste de réels donnant les composantes (4 en 2D et axisymétrie, 6 en 3D) du tenseur des contraintes.

Contraintes thermiques. La classe **SIG_THER** permet de générer des contraintes initiales correspondant à un chargement thermique ou équivalent.

```
ther = SIG_THER(
    ♦ M2 = / 0,
             ♦ VT = vt, [list<float>]
           / 1,
             ♦ NOMF2 = nomf2, [str]
    ♦ TZ = tz, [float]
    ♦ CDIL = cdil, [list<float>]
    ) # fin SIG_THER
```

M2 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier de lecture des températures aux noeuds.
VT : liste de réels donnant les valeurs des températures aux noeuds.
NOMF2 : chaîne de caractères donnant le nom du fichier de lecture des températures aux noeuds.
TZ : réel donnant la température initiale.
CDIL : liste de réels donnant les coefficients de dilatation des groupes d'éléments.

Contraintes géostatiques. La classe **SIG_GEO** permet de générer des contraintes initiales d'origine géostatique. Chaque couche de terrain est définie à l'aide de la classe **SIG_COUC**.

```
geo = SIG_GEO(
    ♦ NC = nc, [int]
    ♦ DEFC = defc, [list<SIG_COUC>]
    ♦ NGO = ngo, [int]
    ◊ NUMG = numg, [list<int>]
    ) # fin SIG_GEO
```

NC : entier donnant le nombre de couches de terrain intervenant pour les contraintes initiales.
NGO : entier donnant le nombre de groupes d'éléments à contraintes initiales nulles.
NUMG : liste d'entiers donnant les numéros des groupes d'éléments à contraintes initiales nulles.

Couches de terrain. La classe **SIG_COUC** permet de décrire chaque couche de terrain.

```
couc = SIG_COUC(
    ♦ HSUP = hsup, [float]
    ♦ PV = pv, [float]
    ♦ CPL1 = cpl1, [float]
    ♦ CPL2 = cpl2, [float]
    ) # fin SIG_COUC
```

HSUP : réel donnant la cote de la limie supérieure de la couche.
PV : réel donnant le poids volumique de la couche.
CPL1 : réel donnant le coefficient de poussée latérale selon Ox.
CPL2 : réel donnant le coefficient de poussée latérale selon Oz (déformation plane) ou Oy (tridimensionnel).

Contraintes isotropes. La classe **SIG_ISO** permet de générer des contraintes initiales de type pression isotrope.

```
iso = SIG_ISO(
    ♦ VERTI = verti, [list<float>]
    ♦ M3 = / 0,
             ♦ VH1 = vh1, [list<float>]
           / 1,
             ♦ NOMF3 = nomf3, [str]
           / 2,
             ♦ H1 = h1, [float]
    ♦ PVF1 = pvf1, [list<float>]
    ♦ M4 = / 0,
             ♦ VH2 = vh2, [list<float>]
           / 1,
             ♦ NOMF4 = nomf4, [str]
           / 2,
             ♦ H2 = h2, [float]
    ) # fin SIG_ISO
```

VERTI : liste de 3 réels donnant les composantes du vecteur unitaire orienté suivant la verticale ascendante.
M3 : entier indiquant le mode de donnée du champ de charge hydraulique initiale à choisir parmi :
— **0** : si on le définit dans le jeu de données,
— **1** : si on le lit dans un fichier de résultats CESAR,
— **2** : si la charge est uniforme.
VH1 : liste de réels donnant les charges hydrauliques aux noeuds.
NOMF3 : chaîne de caractères donnant le nom du fichier de résultats CESAR.
H1 : réel donnant la charge hydraulique.
PVF1 : liste de réels donnant le poids volumique de fluide dans les groupes d'éléments.
M4 : entier indiquant le mode de donnée du champ de charge hydraulique actuelle à choisir parmi :
— **0** : si on le définit dans le jeu de données,
— **1** : si on le lit dans un fichier de résultats CESAR,
— **2** : si la charge est uniforme.
VH2 : liste de réels donnant les charges hydrauliques aux noeuds.
NOMF4 : chaîne de caractères donnant le nom du fichier de résultats CESAR.
H2 : réel donnant la charge hydraulique.

3.27.18 Classe CHAR_SOL

La classe **CHAR_SOL** permet de définir un chargement de type sollicitations nodales.

```
sol = CHAR_SOL(
      ♦ M1 = / 0,
               ♦ GEN = gen, [list<GEN_SOL>]
             / 1,
               ♦ NOMF = nomf, [str]
      ) # fin CHAR_SOL
```

M1 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier de lecture des sollicitations concentrées.
GEN : liste d'objets de type **GEN_SOL** permettant de définir les sollicitations concentrées.
NOMF : chaîne de caratcère donnant le nom du fichier de lecture des sollicitations concentrées.

Génération des noeuds. La classe **GEN_SOL** permet de définir les noeuds soumis aux sollicitations nodales.

```
gen_sol = GEN_SOL(
          ♦ IGEN = / 1,
                     ♦ ID = id, [int]
                     ♦ IF = if, [int]
                     ♦ IPAS = ipas, [int]
                   / 2,
                     ♦ NP = np, [int]
                     ♦ NUM = num, [list<int>]
          ♦ IL = il, [int]
          ♦ F = f, [list<float>]
          ) # fin GEN_SOL
```

IGEN : entier indiquant le mode de génération de l'ensemble des noeuds :
— **1** : pour une génération par **ID**, **IF** et **IPAS**,
— **2** : pour une géénration par **NP** et **NUM**.
ID : entier donnant le premier numéro à générer.
IF : entier donnant le dernier numéro à générer.
IPAS : entier donnant le pas de progression entre **ID** et **IF**.
NP : entier donnant le nombre de numéros de noeuds à définir.
NUM : liste d'entiers donnant les **NP** numéros des noeuds.
IL : entier donnant le numéro du ddl suivant lequel on veut imposer des sollicitations concentrées.
F : liste de réels donnant les valeurs des **NP** sollicitations concentrées.

3.28 Données relatives au type de calcul

3.28.1 Classe AXIF

La classe **AXIF** permet de définir les données pour la résolution d'un problème de mécanique pour les structures élastiques à géométrie de révolution soumises à un chargement quelconque.

```
axif = AXIF(
      ♦ M = / 0,
            / 1,
      ♦ NORDRE = nordre, [int]
      ♦ NPLAN = nplan, [int]
      ◊ # si NPLAN ≠ 0
        ♦ ANGL = angl, [list<float>]
      ♦ IREP = / 0,
                 ♦ ISTK = / 0,
                          / 1,
                            ♦ NOMFS = nomfs, [str]
                 ♦ TAB = tab, [list <AX_CHAR>]
               / 1,
                 ♦ NOMFR = nomfr, [str]
      ) # fin AXIF
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**.
NORDRE : entier indiquant l'ordre de la plus grande des harmoniques utilisées.
NPLAN : entier indiquant le nombre de plans méridiens pour lesquels les résultats doivent être imprimés.
ANGL : liste de réels donnant les valeurs en degrés des angles des plans méridiens.
IREP : entier indiquant s'il s'agit d'un nouveau calcul (**0**) ou d'une reprise (**1**).
ISTK : entier indiquant le stockage (**1**) ou non (**0**) des résultats du calcul pour reprise ultérieure.
NOMFS : chaîne de caractères donnant le nom du fichier de stockage.
NOMFR : chaîne de caractères donnant le nom du fichier de reprise.
TAB : liste d'objets de type **AX_CHAR** permettant de décrire la décompostion en série de Fourier des chargements.

Décomposition en série de Fourier. La classe **AX_CHAR** permet de décrire la décomposition d'un chargement en série de Fourier.

```
char = AX_CHAR(
       ♦ MICD = / 0,
                  ♦ AB = ab, [list <float>]
                / 1,
                  ♦ NTETA = / 2,
                            / 4,
                            / 8,
                            / 16,
                            / 32,
                            / 64,
                            / 128,
                            / 256,
                  ♦ G = g, [list <float>]
       ) # fin AX_CHAR
```

MICD : entier indiquant le mode de description du chargement à choisir parmi :

0 : données des composantes du chargement sur la base des $cos(n\theta)$, $sin(n\theta)$,
1 : données des valeurs du chargement sur un certain nombre de points de discrétisation.

AB : liste de réels donnant les valeurs A_0, A_n, B_n des composantes du chargement sur la base des $cos(n\theta)$, $sin(n\theta)$.
NTETA : entier indiquant le nombre de points de discrétisation du chargement à choisir parmi **2**, **4**, **8**, **16**, **32**, **64**, **128**, **256**.
G : liste de réels donnant les valeurs du chargement aux points de discrétisation.

3.28.2 Classe DTLI

La classe **DTLI** permet de définir les données pour la résolution d'un problème de diffusion transitoire linéaire par intégration directe.

```
dtli = DTLI(
    ♦ M = m, [list <| 0
                    | 1
                    | 2
                    | 4
                    | 8
                 >] # fin list
    ♦ NPAS1 = npas1, [int]
    ♦ T0 = t0, [float]
    ♦ DT = dt, [list<float>]
    ◊ OPT = opt, [list <| CFT
                        | INI
                        | SRE
                        | STK
                        | LIM
                      >] # fin list
    ) # fin DTLI
```

M : liste d'entiers indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**, **4**, **8**.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : liste de réels donnant les valeurs des pas de temps.
OPT : liste d'objets de type à choisir parmi :

- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **INI** pour un calcul avec initialisation de paramètres,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
- **STK** pour un calcul avec stockage des résultats pour reprise de calcul,
- **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,

permettant de définir des options de calcul.

3.28.3 Classe DTNL

La classe **DTNL** permet de définir les données pour la résolution d'un problème de diffusion transitoire non linéaire.

```
dtnl = DTNL(
       ♦ M = m, [list <| 0
                       | 1
                       | 2
                       | 4
                       | 8
                     >] # fin list
       ♦ METHOD = / 'P',
                  / 'N',
       ♦ NITER = niter, [int]
       ♦ TOL = tol, [float]
       ♦ NENL = nenl, [int]
       ♦ NPAS1 = npas1, [int]
       ♦ T0 = t0, [float]
       ♦ DT = dt, [list<float>]
       ◊ OPT = opt, [list <| CFT
                           | INI
                           | SRE
                           | MUL
                           | STK
                           | LIM
                           | ENL
                         >] # fin list
       ) # fin DTNL
```

M : liste d'entiers indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**, **4**, **8**.
METHOD : chaîne de caractère indiquant le type de méthode utilisée à choisir parmi :
— **'P'** : point fixe,
— **'N'** : Newton.
NITER : entier donnant le nombre maximum d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
NENL : entier donnant le nombre de groupes d'éléments d'échange non linéaires.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : liste de réels donnant les valeurs des pas de temps.
OPT : liste d'objets de type à choisir parmi :

- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **INI** pour un calcul avec initialisation de paramètres,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
- **MUL** pour un calcul avec la méthode multifrontale,
- **STK** pour un calcul avec stockage des résultats pour reprise de calcul,
- **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,
- **ENL** pour un calcul avec définition des fonctions du temps relatives aux variations du paramètre extérieur le long d'éléments d'échange non linéaires,

permettant de définir des options de calcul.

3.28.4 Classe DYNI

La classe **DYNI** permet de définir les données pour la résolution d'un problème de recherche de la réponse à une sollicitation dynamique par intégration directe.

```
dyni = DYNI(
        ♦ MID = / 0,
                / 1,
        ♦ MIR = / 0,
                / 1,
                / 2,
        ♦ NPAS1 = npas1, [int]
        ♦ T0 = t0, [float]
        ♦ DT = dt, [float]
        ◊ OPT = opt, [list <| MUL
                           | SRE
                           | STK
                           | AMO
                           | CFT
                           | LIM
                         >] # fin list
        ) # fin DYNI
```

MID : entier indiquant le niveau d'impression des données à choisir parmi **0**, **1**.
MIR : entier indiquant le niveau d'impression des résultats à choisir parmi **0**, **1**, **2**.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : réel donnant la valeur du pas de temps.
OPT : liste d'objets de type à choisir parmi :

- **MUL** pour un calcul avec la méthode multifrontale,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
- **STK** pour un calcul avec stockage des résultats,
- **AMO** pour un calcul avec amortissement,
- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **LIM** pour un calcul avec avec définition des fonctions du temps relatives aux conditions limites,

permettant de définir des options de calcul.

3.28.5 Classe FLAM

La classe **FLAM** permet de définir les données pour la résolution d'un problème de flambement linéaire.

```
flam = FLAM(
        ♦ M = / 0,
              / 1,
        ♦ VD = vd, [float]
        ♦ NVALP = nvalp, [int]
        ♦ NSE = nse, [int]
```

♦ **TOL** = tol, [**float**]
♦ **NITER** = niter, [**int**]
♦ **IVERIF** = / **0**,
/ **1**,
◊ **OPT** = opt, [**list** <| **MUL**
>] # *fin list*
) # *fin FLAM*

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
VD : réel donnant la valeur du shift centrant la recherche des valeurs propres.
NVALP : entier donnant le nombre de valeurs propres.
NSE : entier donnant la dimension du sous-espace.
TOL : réel donnant la précision relative sur les valeurs propres.
NITER : entier donnant le nombre maximum d'itérations de sous-espaces.
IVERIF : entier spécifiant la vérification (**1**) ou non (**0**) du nombre de valeurs propres.
OPT : liste d'objets de type à choisir parmi :
· **MUL** pour un calcul avec la méthode multifrontale,
permettant de spécifier des options de calcul.

3.28.6 Classe LINC

La classe **LINC** permet de définir les données pour la résolution d'un problème de recherche de la réponse à une sollicitation harmonique avec amortissement.

linc = **LINC**(
♦ **OMEGA** = omega, [**float**]
♦ **A** = a, [**float**]
♦ **B** = b, [**float**]
) # *fin LINC*

OMEGA : réel donnant la pulsation de la sollicitation.
A : réel donnant le coefficient de Rayleigh relatif à la rigidité.
B : réel donnant le coefficient de Rayleigh relatif à la masse.

3.28.7 Classe LINE

La classe **LINE** permet de définir les données pour la résolution d'un problème linéaire.

line = **LINE**(
♦ **M** = / **0**,
/ **1**,
♦ **IRG** = / **0**,
/ **1**,
♦ **ISG** = / **0**,
/ **1**,
◊ ♦ **NOMFD** = nomfd, [**str**]
♦ **NOMFS** = nomfs, [**str**]
◊ **OPT** = opt, [**list** <| **MUL**

>] # *fin list*
) # *fin LINE*

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**.
IRG : entier indiquant le caractère nouveau (**0**) ou de reprise sur fichier (**1**) du calcul.
ISG : entier indiquant le stockage (**1**) ou non (**0**) sur fichier de la matrice de rigidité.
NOMFD : chaîne de caractères donnant le nom du fichier contenant la diagonale de la matrice de rigidité.
NOMFS : chaîne de caractères donnant le nom du fichier contenant la partie supérieure la matrice de rigidité.
OPT : liste d'objets de type à choisir parmi :
· **MUL** pour un calcul avec la méthode multifrontale,
permettant de spécifier des options de calcul.

3.28.8 Classe LINH

La classe **LINH** permet de définir les données pour la résolution d'un problème de recherche de la réponse à une sollicitation harmonique sans amortissement.

linh = **LINH**(
♦ **M** = / **0**,
/ **1**,
♦ **OMEGA** = omega, [**float**]
) # *fin LINH*

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**.
OMEGA : réel donnant la pulsation de la sollicitation.

3.28.9 Classe MCNL

La classe **MCNL** permet de définir les données pour la résolution d'un problème mécanique à comportement non linéaire.

mcnl = **MCNL**(
♦ **M** = / **0**,
/ **1**,
/ **2**,
♦ **NINCR** = nincr, [**int**]
♦ **NITER** = niter, [**int**]
♦ **TOL** = tol, [**float**]
♦ **IMET** = / **1**,
/ **2**,
/ **3**,
/ **4**,
/ **11**,
/ **12**,
/ **13**,
/ **21**,

```
                / 22,
                / 23,
◊ VFT = vcorg, [list<float>]
◊ VCT = vcorg, [list<float>]
◊ OPT = opt, [list <| MUL
                   | INI
                   | STK
                   | EST
                   | NDP
                   | CMA
                   | CR1
                   | CR2
                   | CRG
                   | DPL
                   | DTO
                   | FSC
                   | FSR
                   | HPE
                 >] # fin list
) # fin MCNL
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
NINCR : entier donnant le nombre d'incréments.
NITER : entier donnant le nombre d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
IMET : entier indiquant le choix de la méthode de résolution :
— **1** : contraintes initiales,
— **2** : rigidité tangente,
— **3** : mixte avec tangente pour les 2 premières itérations,
— **4** : mixte avec tangente pour les 3 premières itérations,
— **11** : contraintes initiales + line search,
— **12** : contraintes initiales + sécante,
— **13** : contraintes initiales + D-F-P modifiée,
— **21** : contraintes initiales + incrémentation automatique,
— **22** : contraintes initiales + incrémentation automatique + line search,
— **23** : contraintes initiales + incrémentation automatique + sécante.
VFT : liste de réels donnant les coefficients multiplicateurs pour chaque chargement et chaque incrément.
VCT : liste de réels donnant les coefficients multiplicateurs pour chaque condition limite et chaque incrément.
OPT : liste d'objets de type à choisir parmi :
· **MUL** pour un calcul avec la méthode multifrontale,
· **INI** pour un calcul avec initialisation de paramètres,
· **STK** pour un calcul avec stockage des résultats,
· **EST** pour un calcul avec estimation d'erreur a posteriori,
· **NDP** pour un calcul avec annulation des déplacements au moment de la reprise,
· **CMA** pour un calcul avec stockage des contraintes dans la phase matrice (modèle de de Buhan et Sudret),
· **CR1** pour un calcul avec stockage des contraintes dans la phase de renforcement 1

(modèle de de Buhan et Sudret),
- **CR2** pour un calcul avec stockage des contraintes dans la phase de renforcement 2 (modèle de de Buhan et Sudret),
- **CRG** pour un calcul avec stockage des contraintes globales dans les phases de renforcement (modèle de de Buhan et Sudret),
- **DPL** pour un calcul avec stockage des déformations plastiques,
- **DTO** pour un calcul avec stockage des déformations totales,
- **FSC** pour un calcul de facteur de sécurité sur un chargement,
- **FSR** pour un calcul de facteur de sécurité sur les caractéristiques de résistance,
- **HPE** pour un calcul d'homogénéisation périodique,

permettant de définir des options de calcul.

3.28.10 Classe MEXO

La classe **MEXO** permet de définir les données pour la résolution d'un problème de calcul de contraintes dans le béton au jeune âge.

```
mexo = MEXO(
    ♦ M = m, [list <| 0
                    | 1
                    | 2
                    | 4
                  >] # fin list
    ♦ NPAS1 = npas1, [int]
    ◊ OPT = opt, [list <| CFT
                        | EFN
                        | INI
                        | LIM
                        | MUL
                        | PTX
                        | SRE
                        | STK
                        | TXO
                      >] # fin list
    ) # fin MEXO
```

M : liste d'entiers indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**, **4**.
NPAS1 : entier donnant le nombre de pas de temps + 1.
OPT : liste d'objets de type à choisir parmi :
- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **EFN** pour un calcul avec détermination de l'état de déformation dans la direction normale au maillage,
- **INI** pour un calcul avec initialisation de paramètres,
- **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,
- **MUL** pour un calcul avec la méthode multifrontale,
- **PTX** pour un calcul avec lecture des champs de température et de degré d'hydratation sur fichier,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,

· **STK** pour un calcul avec stockage des résultats pour reprise de calcul,
· **TXO** pour un calcul avec lecture des champs de température et de degré d'hydratation sur un fichier de résultats,
permettant de définir des options de calcul.

3.28.11 Classe MODE

La classe **MODE** permet de définir les données pour la résolution d'un problème de recherche de modes propres.

```
mode = MODE(
        ♦ M = / 0,
              / 1,
              / 2,
        ♦ INDIC = / 1,
                  / 2,
                    ♦ NVALP = nvalp, [int]
                    ♦ NSE = nse, [int]
                    ♦ TOL = tol, [float]
                    ♦ NITER = niter, [int]
                    ♦ IVERIF = / 0,
                               / 1,
                    ♦ ISV = / 0,
                            / 1,
                              ♦ NOMF = nomf, [str]
        ♦ VD = vd, [float]
        ◊ OPT = opt, [list <| MUL
                          >] # fin list
        ) # fin MODE
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
INDIC : entier indiquant le type de calcul à choisir parmi :
— **1** : recherche du nombre de valeurs propres inférieures à une valeur donnée,
— **2** : recherche de valeurs propres proches d'une valeur donnée.
NVALP : entier donnant le nombre de modes propres.
NSE : entier donnant la dimension du sous-espace.
TOL : réel donnant la précision relative sur les valeurs propres.
NITER : entier donnant le nombre maximum d'itérations de sous-espaces.
IVERIF : entier spécifiant la vérification (**1**) ou non (**0**) du nombre de valeurs propres.
ISV : entier indiquant le stockage (**1**) ou non (**0**) sur fichier des éléments propres.
NOMF : chaîne de caractères donnant le nom du fichier de stockage des éléments propres.
VD : réel donnant la valeur intervenant dans le choix spécifié dans **INDIC**.
OPT : liste d'objets de type à choisir parmi :
· **MUL** pour un calcul avec la méthode multifrontale,
permettant de spécifier des options de calcul.

3.28.12 Classe MPLI

La classe **MPLI** permet de définir les données pour la résolution d'un problème d'évolution en thermo-mécanique linéaire des milieu poreux.

```
mpli = MPLI(
        ♦ M = / 0,
              / 1,
              / 2,
        ♦ NPAS1 = npas1, [int]
        ♦ T0 = t0, [float]
        ♦ DT = dt, [list<float>]
        ◊ OPT = opt, [list <| CFT
                           | INP
                           | INT
                           | INU
                           | LIM
                           | SRE
                           | STP
                           | STT
                           | STU
                         >] # fin list
        ) # fin MPLI
```

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : liste de réels donnant les valeurs des pas de temps.
OPT : liste d'objets de type à choisir parmi :

- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **INP** pour un calcul avec initialisation de la pression au temps T0,
- **INT** pour un calcul avec initialisation de la température au temps T0,
- **INU** pour un calcul avec initialisation des déplacements nodaux au temps T0,
- **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
- **STP** pour un calcul avec stockage des pressions sur fichier,
- **STT** pour un calcul avec stockage des températures sur fichier,
- **STU** pour un calcul avec stockage des déplacements sur fichier,

permettant de définir des options de calcul.

3.28.13 Classe MPNL

La classe **MPNL** permet de définir les données pour la résolution d'un problème d'évolution en thermo-mécanique non linéaire des milieu poreux.

```
mpnl = MPNL(
        ♦ M = / 0,
              / 1,
              / 2,
        ♦ NITER = niter, [int]
        ♦ TOL = tol, [float]
        ♦ IMET = / 0,
                 / 1,
```

♦ **NPAS1** = npas1, [**int**]
♦ **T0** = t0, [**float**]
♦ **DT** = dt, [**list**<**float**>]
◊ **OPT** = opt, [**list** <| **CFT**
| **DPL**
| **DTO**
| **INI**
| **INP**
| **INT**
| **INU**
| **LIM**
| **SRE**
| **STK**
| **STP**
| **STT**
| **STU**
>] # *fin list*
) # *fin MPNL*

M : entier indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**.
NITER : entier donnant le nombre maximum d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
IMET : entier indiquant le type de méthode utilisée à choisir parmi :
— **1** : contraintes initiales avec schéma semi-implicite,
— **2** : contraintes initiales avec schéma implicite.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : liste de réels donnant les valeurs des pas de temps.
OPT : liste d'objets de type à choisir parmi :
· **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
· **DPL** pour un calcul avec stockage du tenseur des déformations plastiques sur le fichier de résultats,
· **DTO** pour un calcul avec stockage du tenseur des déformations totales sur le fichier de résultats,
· **INI** pour un calcul avec initialisation de paramètres,
· **INP** pour un calcul avec initialisation de la pression au temps T0,
· **INT** pour un calcul avec initialisation de la température au temps T0,
· **INU** pour un calcul avec initialisation des déplacements nodaux au temps T0,
· **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,
· **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
· **STK** pour un calcul avec stockage des résultats pour reprise de calcul,
· **STP** pour un calcul avec stockage des pressions sur fichier,
· **STT** pour un calcul avec stockage des températures sur fichier,
· **STU** pour un calcul avec stockage des déplacements sur fichier,
permettant de définir des options de calcul.

3.28.14 Classe NSAT

La classe **NSAT** permet de définir les données pour la résolution d'un problème d'écoulement en milieu poreux non saturé.

```
nsat = NSAT(
        ♦ M = / 0,
              / 1,
              / 2,
              / 4,
              / 8,
        ♦ V = v, [list<float>]
        ♦ MET1 = / 'P',
                 / 'N',
        ♦ MET2 = / 'H',
        ♦ NITER = niter, [int]
        ♦ TOL = tol, [float]
        ♦ NPAS1 = npas1, [int]
        ♦ T0 = t0, [float]
        ♦ DT = dt, [float]
        ◊ OPT = opt, [list <| CFT
                           | INI
                           | SRE
                           | STK
                           | LIM
                           | SUI
                         >] # fin list
        ) # fin NSAT
```

M : liste d'entiers indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**, **4**, **8**.
V : liste de réels donnant les composantes dans le repère du maillage du vecteur unitaire orienté suivant la verticale ascendante.
MET1 : chaîne de caractère indiquant le type de méthode utilisée à choisir parmi :
— **'P'** : point fixe,
— **'N'** : Newton.
MET2 : chaîne de caractère indiquant le type de méthode utilisée à choisir parmi :
— **'H'** : résolution à l'aide de la charge hydraulique.
NITER : entier donnant le nombre maximum d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : liste de réels donnant les valeurs des pas de temps.
OPT : liste d'objets de type à choisir parmi :

- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **INI** pour un calcul avec initialisation de paramètres,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
- **STK** pour un calcul avec stockage des résultats pour reprise de calcul,
- **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,

· **SUI** pour un calcul avec définition des surfaces de suintement,
permettant de définir des options de calcul.

3.28.15 Classe SUMO

La classe **SUMO** permet de définir les données pour la résolution d'un problème de recherche de la réponse à une sollicitation dynamique par superposition modale.

```
sumo = SUMO(
        ♦ M = / 0,
              / 1,
              / 2,
        ♦ NOMF = nomf, [str]
        ♦ KMODE = kmode, [list<int>]
        ♦ IPROB = / 1, # calcul spectral SRSS
                    ♦ ISPEC = / 0,
                                ♦ NPAS1 = npas1, [int]
                                ♦ T0 = t0, [float]
                                ♦ DT = dt, [float]
                              / 1,
                                ♦ SPEC = spec, [list<float>]
                              / 2,
                                ♦ SPEC = spec, [list<float>]
                  / 2, # réponse complète
                    ♦ NPAS1 = npas1, [int]
                    ♦ T0 = t0, [float]
                    ♦ DT = dt, [float]
                  / 3, # calcul spectral CQC
                    ♦ ISPEC = / 0,
                                ♦ NPAS1 = npas1, [int]
                                ♦ T0 = t0, [float]
                                ♦ DT = dt, [float]
                              / 1,
                                ♦ SPEC = spec, [list<float>]
                              / 2,
                                ♦ SPEC = spec, [list<float>]
        ◊ OPT = opt, [list <| AMO
                           | CFT
                           | SRE
                           | INI
                         >] # fin list
        ) # fin SUMO
```

M : entier indiquant le niveau d'impression des résultats à choisir parmi **0**, **1**, **2**.
NOMF : chaîne de caractère donnant le nom du fichier de lecture des modes propres.
KMODE : liste d'entiers donnant les numéros des modes prorpes retenus pour la superposition.
IPROB : entier indiquant le type de calcul :
— **1** : spectral SRSS,

— **2** : réponse complète,
— **3** : spectral CQC.
ISPEC : entier indiquant si le spectre de réponse à la sollicitation est donné à choisir parmi :
— **1** : calculé en déplacement,
— **2** : donné en déplacement,
— **3** : donné en accélération.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : réel donnant la valeur du pas de temps.
SPEC : liste de réels de même taille que **KMODE** donnant les valeurs du spectre de réponse aux modes retenus.
OPT : liste d'objets de type à choisir parmi :
· **AMO** pour un calcul avec amortissement,
· **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
· **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
· **INI** pour un calcul avec initialisation des paramètres,
permettant de définir des options de calcul.

3.28.16 Classe SURF

La classe **SURF** permet de définir les données pour la résolution d'un problème d'écoulement plan en milieu poreux avec surfaces libres.

```
surf = SURF(
    ♦ POIDS = poids, [float]
    ♦ VERTI = verti, [list<float>]
    ♦ NITER = niter, [int]
    ♦ TOL = tol, [float]
    ◊ OPT = opt, [list <| STK
                       | SUI
                    >] # fin list
    ) # fin SURF
```

POIDS : réel donnant le poids volumique du fluide.
VERTI : liste de réels donnant les composantes dans le repère du maillage du vecteur unitaire vertical ascendant.
NITER : entier donnant le nombre maximum d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
OPT : liste d'objets de type à choisir parmi :
· **STK** pour un calcul avec stockage sur fichier des charges nodales pour reprise de calcul,
· **SUI** pour un calcul avec définition des surfaces de suintement,
permettant de définir des options de calcul.

3.28.17 Classe TCNL

La classe **TCNL** permet de définir les données pour la résolution d'un problème de contact entre solides à comportement non linéaire.

```
tcnl = TCNL(
        ♦ M = / 0,
              / 1,
              / 2,
              / 3,
        ♦ NINCR = nincr, [int]
        ♦ NITER = niter, [int]
        ♦ TOL = tol, [float]
        ♦ IAUTO = / 0,
        ♦ KNP = / 1,
                / 0,
        ♦ KND = / 1,
                / 0,
        ♦ KNF = / 1,
                / 0,
        ◊ VFT = vft, [list<float>]
        ◊ OPT = opt, [list <| INI
                           | STK
                           | NDP
                           | DPL
                           | MUL
                         >] # fin list
        ) # fin TCNL
```

M : entier indiquant le niveau d'impression des résultats à choisir parmi **0**, **1**, **2**, **3**.
NINCR : entier donnant le nombre d'incréments.
NITER : entier donnant le nombre d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
IAUTO : entier indiquant la méthode de résolution à choisir parmi **0** :
— **0** : incrémentation manuelle.
KNP : entier indiquant si le critère de non interpénétration doit être vérifié (**1**) ou non (**0**).
KND : entier indiquant si le critère de décollement doit être vérifié (**1**) ou non (**0**).
KNF : entier indiquant si le critère de frottement doit être vérifié (**1**) ou non (**0**).
VFT : liste de réels donnant les coefficients multiplicateurs pour chaque chargement et chaque incrément.
OPT : liste d'objets de type à choisir parmi :

- **INI** pour un calcul avec initialisation de paramètres,
- **STK** pour un calcul avec stockage des résultats,
- **NDP** pour un calcul avec annulation des déplacements au moment de la reprise,
- **DPL** pour un calcul avec stockage du tenseur des déformations plastiques sur le fichier de résultats,
- **MUL** pour un calcul avec la méthode multifrontale,

permettant de définir des options de calcul.

3.28.18 Classe TEXO

La classe **TEXO** permet de définir les données pour la résolution d'un problème de calcul de champ de température dans une pièce en béton en cours de prise.

```
texo = TEXO(
        ♦ M = m, [list <| 0
                        | 1
                        | 2
                        | 4
                        | 8
                      >] # fin list
        ♦ NITER = niter, [int]
        ♦ TOL = tol, [float]
        ♦ NPAS1 = npas1, [int]
        ♦ T0 = t0, [float]
        ♦ DT = dt, [float]
        ◊ OPT = opt, [list <| CFT
                           | INA
                           | INI
                           | LIM
                           | PTX
                           | QAB
                           | SRE
                           | STK
                         >] # fin list
        ) # fin TEXO
```

M : liste d'entiers indiquant le niveau d'impression à choisir parmi **0**, **1**, **2**, **4**, **8**.
NITER : entier donnant le nombre maximum d'itérations.
TOL : réel donnant la tolérance relative sur la convergence.
NPAS1 : entier donnant le nombre de pas de temps + 1.
T0 : réel donnant la valeur du temps initial.
DT : liste de réels donnant les valeurs des pas de temps.
OPT : liste d'objets de type à choisir parmi :

- **CFT** pour un calcul avec définition des fonctions du temps relatives aux chargements,
- **INA** pour un calcul avec déclaration des groupes d'éléments inactifs,
- **INI** pour un calcul avec initialisation de paramètres,
- **LIM** pour un calcul avec définition des fonctions du temps relatives aux conditions limites,
- **PTX** pour un calcul avec écriture des champs de température et de degré d'hydratation sur fichier,
- **QAB** pour un calcul avec donnée des résultats de l'essai QAB,
- **SRE** pour un calcul avec stockage des résultats pour exploitation graphique,
- **STK** pour un calcul avec stockage des résultats pour reprise de calcul,

permettant de définir des options de calcul.

3.29 Données relatives aux options de calcul

3.29.1 Classe AMO

La classe **AMO** permet de définir les données pour la prise en compte d'un amortissement.

```
amo = AMO(
```

```
  ♦ MOD = / 'DYNI',
            ♦ A = a, [float]
            ♦ B = b, [float]
          / 'SUMO',
            ♦ XI = xi, [list<float>]
  ) # fin AMO
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'DYNI'**, **'SUMO'** auquel s'applique l'option.
A : réel donnant le coefficient de Rayleigh relatif à la matrice de rigidité.
B : réel donnant le coefficient de Rayleigh relatif à la matrice de masse.
XI : liste de réels donnant les pourcentages d'amortissement critique associés aux modes retenus.

3.29.2 Classe CFT

La classe **CFT** permet de définir les fonctions du temps relatives aux chargements.

```
cft = CFT(
    ♦ MOD = / 'DYNI',
            / 'SUMO',
            / 'DTLI',
            / 'DTNL',
            / 'MEXO',
            / 'MPLI',
            / 'MPNL',
            / 'NSAT',
            / 'TEXO',
    ♦ / # si 'DYNI', 'SUMO'
      ♦ M2 = / 0,
               ♦ FT = ft, [list<float>]
             / 1,
               ♦ NOMF = nomf, [str]
      / # si 'DTLI', 'DTNL', 'MEXO', 'MPLI', 'MPNL', 'NSAT', 'TEXO'
      ♦ FT = ft, [list<float>]
    ) # fin CFT
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'DYNI'**, **'SUMO'**, **'DTLI'**, **'DTNL'**, **'MEXO'**, **'MPLI'**, **'MPNL'**, **'NSAT'**, **'TEXO'** auquel s'applique l'option.
M2 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier de lecture des fonctions du temps.
FT : liste de réels donnant les valeurs de chaque fonction du temps à chaque pas de temps.
NOMF : chaîne de caractères donnant le nom du fichier de lecture des fonctions du temps.

3.29.3 Classe CMA

La classe **CMA** permet de définir les données pour le stockage dans le fichier de résultats des contraintes dans la phase matrice du modèle de de Buhan et Sudret.

cma = **CMA**(
♦ **MOD** = / **'MCNL'**,
) # *fin CMA*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'** auquel s'applique l'option.

3.29.4 Classe CR1

La classe **CR1** permet de définir les données pour le stockage dans le fichier de résultats des contraintes dans la phase de renforcement 1 du modèle de de Buhan et Sudret.

cr1 = **CR1**(
♦ **MOD** = / **'MCNL'**,
) # *fin CR1*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'** auquel s'applique l'option.

3.29.5 Classe CR2

La classe **CR2** permet de définir les données pour le stockage dans le fichier de résultats des contraintes dans la phase de renforcement 2 du modèle de de Buhan et Sudret.

cr2 = **CR2**(
♦ **MOD** = / **'MCNL'**,
) # *fin CR2*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'** auquel s'applique l'option.

3.29.6 Classe CRG

La classe **CRG** permet de définir les données pour le stockage dans le fichier de résultats des contraintes globales dans les phases de renforcement du modèle de de Buhan et Sudret.

crg = **CRG**(
♦ **MOD** = / **'MCNL'**,
) # *fin CRG*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'** auquel s'applique l'option.

3.29.7 Classe DPL

La classe **DPL** permet de définir les données pour le stockage sur le fichier de résultats du tenseur des déformations plastiques.

dpl = **DPL**(
♦ **MOD** = / **'MCNL'**,
/ **'TCNL'**,
/ **'MPNL'**,
) # *fin DPL*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, **'TCNL'**, **'MPNL'** auquel s'applique l'option.

3.29.8 Classe DTO

La classe **DTO** permet de définir les données pour le stockage sur le fichier de résultats du tenseur des déformations totales.

dto = **DTO**(
♦ **MOD** = / **'MCNL'**,
/ **'TCNL'**,
) # *fin DTO*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, **'TCNL'** auquel s'applique l'option.

3.29.9 Classe EFN

La classe **EFN** permet de définir les données pour le calcul de l'état de déformation dans la direction normale au maillage, de façon à ce que le torseur des efforts extérieurs imposés dans cette direction soit nul.

efn = **EFN**(
♦ **MOD** = / **'MEXO'**,
) # *fin EFN*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MEXO'** auquel s'applique l'option.

3.29.10 Classe ENL

La classe **ENL** permet de définir les données pour les fonctions du temps relatives aux variations du paramètre extérieur le long des éléments d'échange non linéaires.

enl = **ENL**(
♦ **MOD** = / **'DTNL'**,
♦ **VPE** = vpe, [**list** <**VPEG**>]
) # *fin ENL*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'DTNL'** auquel s'applique l'option.
VPE : liste d'objets de type **VPEG** décrivant pour chaque groupe d'éléments d'échange non linéaire la fonction du temps relative aux variations du paramètre extérieur le long du groupe.

Fonction du temps relative aux variations du paramètre extérieur. La classe **VPEG** permet de décrire une fonction du temps relative aux variations du paramètre extérieur le long d'un groupe d'éléments d'échange non linéaire.

vpeg = **VPEG**(
♦ **IG** = ig, [**int**]
♦ **EXT** = ext, [**list**<**float**>]
) # *fin VPEG*

IG : entier donnant le numéro du groupe.
EXT : liste de réels donnant la valeur initiale et à chaque pas de temps du paramètre extérieur en bordure du groupe.

3.29.11 Classe EST

La classe **EST** permet de définir les données pour un calcul d'estimation d'erreur. Cette option est pour l'instant limitée aux modèles mécaniques en élasticité linéaire plane.

est = **EST**(
♦ **MOD** = / **'MCNL'**,
♦ **ITYPEE** = / **1**,
) # *fin EST*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, auquel s'applique l'option.
ITYPEE : indicateur spécifiant le type d'estimateur d'erreur à utiliser :
— **1** : pour l'estimateur au sens de Zhu-Zienkiewicz (version 1992 dite "ZZ2", lissage par patch d'éléments).

3.29.12 Classe FSC

La classe **FSC** permet de définir les données pour un calcul de facteur de sécurité sur un chargement (recherche automatique de charge limite).

fsc = **FSC**(
♦ **MOD** = / **'MCNL'**,
♦ **IFC** = / **0**,
/ **1**,
♦ **N** = n, [**int**]
♦ **RMIN** = rmin, [**float**]
♦ **RMAX** = rmax, [**float**]
♦ **PREC** = prec, [**float**]
) # *fin FSC*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, auquel s'applique l'option.
IFC : indicateur entier :
— **0** : si on fait le nombre maximum d'itérations **NITER** spécifiées dans **MCNL**,
— **1** : si on tente de détecter les non-convergences en cours d'analyse.
N : entier donnant le numéro du chargement à optimiser.
RMIN : réel donnant la valeur minimale de l'intervalle de recherche pour le facteur de sécurité
RMAX : réel donnant la valeur maximale de l'intervalle de recherche pour le facteur de sécurité.
PREC : réel donnant la précision souhaitée sur la valeur du facteur de sécurité.

3.29.13 Classe FSR

La classe **FSR** permet de définir les données pour un calcul de facteur de sécurité sur les caractéristiques de résistance (c-phi réduction).

fsr = **FSR**(
♦ **MOD** = / **'MCNL'**,
♦ **IFC** = / **0**,
/ **1**,
♦ **VMIN** = vmin, [**float**]
♦ **VMAX** = vmax, [**float**]
♦ **PREC** = prec, [**float**]
) # *fin FSR*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, auquel s'applique l'option.
IFC : indicateur entier :
— **0** : si on fait le nombre maximum d'itérations **NITER** spécifiées dans **MCNL**,
— **1** : si on tente de détecter les non-convergences en cours d'analyse.
VMIN : réel donnant la valeur minimale de l'intervalle de recherche pour le facteur de sécurité
VMAX : réel donnant la valeur maximale de l'intervalle de recherche pour le facteur de sécurité.
PREC : réel donnant la précision souhaitée sur la valeur du facteur de sécurité.

3.29.14 Classe HPE

La classe **HPE** permet de réaliser des calculs d'homogénéisation périodique en post-traitement.

hpe = **HPE**(
♦ **MOD** = / **'MCNL'**,
) # *fin HPE*

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'** auquel s'applique l'option.

3.29.15 Classe INA

La classe **INA** permet de définir les données pour la déclaration de groupes inactifs.

```
ina = INA(
      ♦ MOD = / 'TEXO',
      ♦ NG = ng, [int]
      ♦ KG = kg, [list<int>]
      ) # fin INA
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'TEXO'**, auquel s'applique l'option.
NG : entier donnant le nombre de groupes inactifs.
KG : liste d'entiers donnant les numéros des groupes inactifs.

3.29.16 Classe INI

La classe **INI** permet de définir les données pour l'initialisation de champs de paramètres en début de calcul.

```
ini = INI(
      ♦ MOD = / 'MCNL',
              / 'TCNL',
              / 'DYNI',
              / 'SUMO',
              / 'DTNL',
              / 'DTLI',
              / 'NSAT',
              / 'TEXO',
              / 'MEXO',
              / 'MPNL',
      ♦ / # si 'MCNL', 'TCNL', 'MEXO', 'MPNL'
          ♦ NOMF = nomf, [str]
        / # si 'DYNI', 'SUMO'
          ♦ M2 = / 0,
                   ♦ VUI = vui, [list<float>]
                   ♦ VVI = vvi, [list<float>]
                 / 1,
                   ♦ NOMF = nomf, [str]
        / # si 'DTNL', 'NSAT'
          ♦ IPERM = / 0,
                    / 1,
          ♦ M2 = / 0,
                   ♦ VU = vu, [list<float>]
                 / 1,
                   ♦ VU0 = vu0, [float]
                 / 2,
                   ♦ NG = ng, [int]
                   ♦ KG_VU0 = kg_vu0, [list<float>]
```

```
                / 3,
                  ♦ NOMF = nomf, [str]
  / # si 'DTLI'
    ♦ IPERM = / 0,
                    ♦ M2 = / 0,
                             ♦ VU = vu, [list<float>]
                           / 1,
                             ♦ VU0 = vu0, [float]
                           / 2,
                             ♦ NG = ng, [int]
                             ♦ KG_VU0 = kg_vu0, [list<int, float>]
                           / 3,
                             ♦ NOMF = nomf, [str]
                  / 1,
  / # si 'TEXO'
    ♦ M2 = / 0,
             ♦ VU = vu, [list<float>]
           / 1,
             ♦ VU0 = vu0, [float]
           / 2,
             ♦ NG = ng, [int]
             ♦ KG_TG = kg_tg, [list<int, float>]
           / 3,
             ♦ NOMF = nomf, [str]
           / 4,
             ♦ NOMF = nomf, [str]
) # fin INI
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, **'TCNL'**, **'DYNI'**, **'SUMO'**, **'SUMO'**, **'DTNL'**, **'DTLI'**, **'NSAT'**, **'TEXO'**, **'MEXO'**, **'MPNL'** auquel s'applique l'option.

 # si 'MCNL', 'TCNL', 'MEXO', 'MPNL'
NOMF : chaîne de caractères donnant le nom du fichier de reprise.

 # si 'DYNI', 'SUMO'
M2 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier de lecture des déplacements et vitesses initiaux.
VUI : liste de réels donnant les déplacements initiaux.
VVI : liste de réels donnant les vitesses initiales.
NOMF : chaîne de caractères donnant le nom du fichier de lecture des déplacements et vitesses initiaux.

 # si 'DTNL', 'NSAT', 'DTLI'
IPERM : entier indiquant si le champ de paramètre est initialisé par un calcul en régime permanent (**1**) ou par un autre procédé (**0**).
M2 : entier indiquant si le champ de paramètre est initialisé par lecture de tableaux de valeurs (**0**), à une même valeur en tout noeud (**1**), par groupe (**2**) ou par lecture sur un fichier de reprise (**3**).

VU : liste de réels donnant le champ initial des paramètres nodaux.
VU0 : réel donnant la valeur initiale du paramètre.
NG : entier donnant le nombre de groupes d'éléments de massif pour lesquels on initialise le champ de paramètre.
KG_VU0 : liste de couples entier, réel donnant pour chaque couple le numéro du groupe et la valeur initiale du paramètre sur ce groupe.
NOMF : chaîne de caractères donnant le nom du fichier de lecture du champ initial des paramètres nodaux.

 # si 'TEXO'
M2 : entier indiquant si le champ de température est initialisé par lecture de tableaux de valeurs (**0**), à une même valeur en tout noeud (**1**), par groupe (**2**),par lecture sur un fichier avec (**3**) ou sans (**4**) reprise des quantités de chaleur d'hydratation.
VU : liste de réels donnant le champ initial des températures nodales.
VU0 : réel donnant la valeur initiale de la température.
NG : entier donnant le nombre de groupes d'éléments de massif pour lesquels on initialise la température.
KG_TG : liste de couples entier, réel donnant pour chaque couple le numéro du groupe et la valeur initiale de la température sur ce groupe.
NOMF : chaîne de caractères donnant le nom du fichier de lecture du champ initial des températures nodales.

3.29.17 Classe INP

La classe **INP** permet de définir les données pour l'initialisation de la pression au temps T0.

```
inp = INP(
      ♦ MOD = / 'MPNL',
                / 'MPLI',
      ♦ M4 = / 0,
                ♦ P = p, [list<float>]
              / 1,
                ♦ P0 = p0, [float]
              / 2,
                ♦ Z0 = z0, [float]
                ♦ PZ0 = pz0, [float]
                ♦ GP = gp, [float]
              / 3,
                ♦ NOMF = nomf, [str]
      ) # fin INP
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MPNL'**, **'MPLI'** auquel s'applique l'option.
M4 : entier indiquant le mode de données des pressions initiales à choisir parmi :
— **0** : lecture d'un tableau de pressions initiales aux noeuds,
— **1** : initialisation à une même valeur en tout noeud,
— **2** : calcul de la pression en fonction de la cote,
— **3** : lecture sur un fichier de reprise d'un précédent calcul.

P : liste de réels donnant les valeurs des pressions initiales aux noeuds.
P0 : réel donnant la valeur initiale de la pression.
Z0 : réel donnant la valeur de la cote verticale au-dessus de laquelle la pression est nulle.
PZ0 : réel donnant la valeur de la pression à la cote verticale **Z0**.
GP : réel donnant le gradient vertical de la pression.
NOMF : chaîne de caractères donnant le nom du fichier sur lequel est lu le tableau des pressions initiales.

3.29.18 Classe INT

La classe **INT** permet de définir les données pour l'initialisation de la température au temps T0.

```
int = INT(
      ♦ MOD = / 'MPNL',
              / 'MPLI',
      ♦ M5 = / 0,
               ♦ T = t, [list<float>]
             / 1,
               ♦ T0 = t0, [float]
             / 3,
               ♦ NOMF = nomf, [str]
      ) # fin INT
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MPNL'**, **'MPLI'** auquel s'applique l'option.
M5 : entier indiquant le mode de données des températures initiales à choisir parmi :
— **0** : lecture d'un tableau de températures initiales aux noeuds,
— **1** : initialisation à une même valeur en tout noeud,
— **3** : lecture sur un fichier de reprise d'un précédent calcul.
T : liste de réels donnant les valeurs des températures initiales aux noeuds.
T0 : réel donnant la valeur initiale de la température.
NOMF : chaîne de caractères donnant le nom du fichier sur lequel est lu le tableau des températures initiales.

3.29.19 Classe INU

La classe **INU** permet de définir les données pour l'initialisation des déplacements au temps T0.

```
inu = INU(
      ♦ MOD = / 'MPNL',
              / 'MPLI',
      ♦ M3 = / 0,
               ♦ U = u, [list<float>]
             / 3,
               ♦ NOMF = nomf, [str]
      ) # fin INU
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MPNL'**, **'MPLI'** auquel s'applique l'option.

M3 : entier indiquant le mode de données des déplacements initiaux à choisir parmi :
— **0** : lecture d'un tableau de déplacements initiaux aux noeuds,
— **3** : lecture sur un fichier de reprise d'un précédent calcul.
U : liste de réels donnant les valeurs des déplacements initiaux aux noeuds.
NOMF : chaîne de caractères donnant le nom du fichier sur lequel est lu le tableau des températures initiales.

3.29.20 Classe LIM

La classe **LIM** permet de définir des conditions limites sur l'inconnue principale variables avec le temps.

```
lim = LIM(
    ♦ MOD = / 'DYNI',
              / 'DTNL',
              / 'DTLI',
              / 'NSAT',
              / 'TEXO',
              / 'MEXO',
              / 'MPNL',
              / 'MPLI',
    ♦ / # si 'DTNL', 'DTLI', 'NSAT', 'TEXO', 'MPNL', 'MPLI'
        ♦ TAB = tab, [list<KVCOND>]
      / # si 'DYNI'
        ♦ M2 = / 0,
                   ♦ TAB = tab, [list<KVCOND>]
                 / 1,
                   ♦ NOMF = nomf, [str]
      / # si 'MEXO'
        ♦ VCT = vct, [list<float>]
    ) # fin LIM
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'DYNI'**, **'DTNL'**, **'DTLI'**, **'NSAT'**, **'TEXO'**, **'MEXO'**, **'MPNL'**, **'MPLI'** auquel s'applique l'option.
M2 : entier indiquant l'utilisation (**1**) ou non (**0**) d'un fichier de lecture des conditions limites fonctions du temps.
TAB : liste d'objets de type **KVCOND** donnant les conditions limites fonctions du temps.
NOMF : chaîne de caractères donnant le nom du fichier de lecture des conditions limites fonctions du temps.
VCT : liste de réels donnant les coefficients multiplicateurs de chaque condition limite pour chaque pas de temps.

Fonctions du temps. La classe **KVCOND** permet de définir une fonction du temps à appliquer à une condition limite.

```
kvcond = KVCOND(
        ♦ KCOND = kcond, [list < | 0
                                 | 1
```

```
                                    >] # fin list
  ♦ VCOND = vcond, [list<float>]
  ) # fin KVCOND
```

KCOND : liste d'entiers indiquant la prise en compte (**1**) ou pas (**0**) de la condition limite pour chaque pas de temps.
VCOND : liste de réels de même taille que **KCOND** donnant les coefficients multiplicateurs de la condition limite pour chaque pas de temps.

3.29.21 Classe MUL

La classe **MUL** permet de définir les données pour l'utilisation de la méthode multifrontale pour la résolution du système linéaire.

```
mul = MUL(
  ♦ MOD = / 'LINE',
          / 'MCNL',
          / 'DYNI',
          / 'DTNL',
          / 'TEXO',
          / 'MEXO',
          / 'TCNL',
  ♦ IGND = / 0,
           / 1,
  ♦ IIO = / 0,
          / 1,
  ) # fin MUL
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'LINE'**, **'MCNL'**, **'DYNI'**, **'DTNL'**, **'TEXO'**, **'MEXO'**, **'TCNL'** auquel s'applique l'option.
IGND : entier donnant le type d'algorithme à choisir parmi **0**, **1** :
— **0** : degré minimum,
— **1** : dissection emboîtée généralisée.
IIO : entier indiquant si les facteurs sont écrits en mémoire (**0**) ou sur fichier (**1**).

3.29.22 Classe NDP

La classe **NDP** permet de définir les données pour l'annulation des déplacements au moment de la lecture sur un fichier de reprise.

```
ndp = NDP(
  ♦ MOD = / 'MCNL',
          / 'TCNL',
  ) # fin NDP
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, **'TCNL'** auquel s'applique l'option.

3.29.23 Classe PTX

La classe **PTX** permet de définir de définir les données pour l'écriture ou la lecture des champs de température et de degré d'hydratation sur un fichier spécifique.

```
ptx = PTX(
      ♦ MOD = / 'TEXO',
              / 'MEXO',
      ♦ / # si 'MEXO'
        ♦ M0 = / 0,
                 ♦ NOMF = nomf, [str]
               / 1,
                 ♦ KPAS = kpas, [list<int>]
        ) # fin PTX
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'TEXO'**, **'MEXO'** auquel s'applique l'option.
M0 : entier indiquant l'utilisation (**0**) ou non (**1**) de tous les pas de temps stockés lors du calcul TEXO.
NOMF : chaîne de caractères donnant le nom du fichier spécifique (.prtx) créé lors du calcul TEXO.
KPAS : liste d'entiers donnant les numéros des pas de temps stockés lors du calcul TEXO intervenant lors du calcul MEXO.

3.29.24 Classe QAB

La classe **QAB** permet de définir les données pour les résultats de l'essai QAB.

```
qab = QAB(
      ♦ MOD = / 'TEXO',
      ♦ NOP = nop, [int]
      ♦ VALM = valm, [list<float>]
      ♦ A = a, [float]
      ♦ B = b, [float]
      ♦ C = c, [float]
      ♦ CM = cm, [float]
      ♦ XK = xk, [float]
      ) # fin QAB
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'TEXO'** auquel s'applique l'option.
NOP : entier indiquant le nombre de triplets (temps, température échantillon, température extérieur) de l'essai.
VALM : liste de réels donnant les valeurs des triplets (temps, température échantillon, température extérieur) de l'essai.
A : réel donnant la valeur du coefficient caractéristique A des déperditions thermiques du calorimètre.
B : réel donnant la valeur du coefficient caractéristique B des déperditions thermiques du calorimètre.

C : réel donnant la valeur du coefficient caractéristique C des déperditions thermiques du calorimètre.
CM : réel donnant la valeur de la capacité calorifique de l'échantillon.
XK : réel donnant la valeur de la constante de la loi d'Arrhénius.

3.29.25 Classe SRE

La classe **SRE** permet de définir les données pour le stockage des résultats en vue de leur exploitation graphique.

```
sre = SRE(
      ♦ MOD = / 'DYNI',
              / 'SUMO',
              / 'DTNL',
              / 'DTLI',
              / 'NSAT',
              / 'TEXO',
              / 'MEXO',
              / 'MPNL',
              / 'MPLI',
      ♦ KSRE = ksre, [list<int>]
      ♦ ISRC = / 0,
              / 1,
              / 2,
      ) # fin SRE
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'DYNI'**, **'SUMO'**, **'DTNL'**, **'DTLI'**, **'NSAT'**, **'TEXO'**, **'MEXO'**, **'MPNL'**, **'MPLI'** auquel s'applique l'option.
KSRE : liste d'indicateurs spécifiant pour chaque pas de temps si les résultats sont stockés (**1**) ou pas (**0**).
ISRC : entier indiquant le niveau de stockage souhaité :
— **0** : pour le stockage de l'inconnue principale,
— **1** : pour le stockage de l'inconnue principale et des résultats complémentaires,
— **2** : dans le cas du module **DYNI** uniquement, pour un stockage identique au cas **1** mais incluant les déformations dans les résultats complémentaires.

3.29.26 Classe STK

La classe **STK** permet de définir les données pour le stockage des résultats du calcul pour reprise ultérieure.

```
stk = STK(
      ♦ MOD = / 'MCNL',
              / 'TCNL',
              / 'DYNI',
              / 'DTNL',
              / 'DTLI',
              / 'NSAT',
```

```
                / 'TEXO',
                / 'MEXO',
                / 'MPNL',
                / 'SURF',
    ♦ NOMF = nomf, [str]
    ) # fin STK
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MCNL'**, **'TCNL'**, **'DYNI'** , **'DTNL'**, **'DTLI'**, **'NSAT'**, **'TEXO'**, **'MEXO'**, **'MPNL'**, **'SURF'** auquel s'applique l'option.
NOMF : chaîne de caractères donnant le nom du fichier de stockage des résultats.

3.29.27 Classe STP

La classe **STP** permet de définir les données pour le stockage des pressions sur fichier.

```
stp = STP(
    ♦ MOD = / 'MPNL',
            / 'MPLI',
    ♦ NOMF = nomf, [str]
    ) # fin STP
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MPNL'**, **'MPLI'** auquel s'applique l'option.
NOMF : chaîne de caractères donnant le nom du fichier de stockage des résultats.

3.29.28 Classe STT

La classe **STT** permet de définir les données pour le stockage des températures sur fichier.

```
stt = STT(
    ♦ MOD = / 'MPNL',
            / 'MPLI',
    ♦ NOMF = nomf, [str]
    ) # fin STT
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MPNL'**, **'MPLI'** auquel s'applique l'option.
NOMF : chaîne de caractères donnant le nom du fichier de stockage des résultats.

3.29.29 Classe STU

La classe **STU** permet de définir les données pour le stockage des déplacements sur fichier.

```
stu = STU(
    ♦ MOD = / 'MPNL',
            / 'MPLI',
    ♦ NOMF = nomf, [str]
    ) # fin STU
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MPNL'**, **'MPLI'** auquel s'applique l'option.
NOMF : chaîne de caractères donnant le nom du fichier de stockage des résultats.

3.29.30 Classe SUI

La classe **SUI** permet de définir les données pour la définition des surfaces de suintement.

```
sui = SUI(
    ♦ MOD = / 'NSAT',
            / 'SURF',
    ♦ NF = nf, [int]
    ♦ NMF = nmf, [int]
    ♦ KNF = knf, [list<int>]
    ) # fin SUI
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'NSAT'**, **'SURF'** auquel s'applique l'option.
NF : entier donnant le nombre de facettes situées sur les surfaces de suintement.
NMF : entier donnant le nombre maximum de noeuds d'une facette.
KNF : liste d'entiers donnant la numérotation des facettes de suintement.

3.29.31 Classe TXO

La classe **TXO** permet de définir les données pour la lecture des champs de température et de degré d'hydratation sur le fichier de résultats du calcul TEXO.

```
txo = TXO(
    ♦ MOD = / 'MEXO',
    ♦ NOMF = nomf, [str]
    ♦ M0 = / 0,
           / 1,
             ♦ KPAS = kpas, [list<int>]
    ) # fin TXO
```

MOD : chaîne de caractères donnant le nom du module d'exécution de CESAR à choisir parmi **'MEXO'** auquel s'applique l'option.
M0 : entier indiquant l'utilisation (**0**) ou non (**1**) de tous les pas de temps stockés lors du calcul TEXO.
NOMF : chaîne de caractères donnant le nom du fichier de résultats créé par le calcul TEXO.
KPAS : liste d'entiers donnant les numéros des pas de temps stockés lors du calcul TEXO intervenant lors du calcul MEXO.

3.30 Pré/post-traitement

Le langage CESAR permet de retrouver la mise en données "classique" de CESAR (celle du .data) ce qui permet de piloter finement le solveur, mais ce qui présente aussi l'inconvénient

de devoir construire soi-même certaines données fastidieuses, notamment celles relatives au maillage (listes **VCORG**, **NUMEL**, **PNUMEL**, ...). De plus, les résultats du calcul sont disponibles principalement dans le fichier de résultats binaire (.rsv4) difficilement exploitable par l'utilisateur (noter cependant la présence du module **EXPO** permettant d'exporter le maillage et les résultats au format GMSH).
C'est en effet au niveau du langage Pilote (de plus haut niveau d'abstraction) que l'on propose de façon naturelle des facilités de pré/post-traitement.
Néanmoins, afin d'améliorer la situation pour l'utilisateur du langage CESAR, 2 classes supplémentaires ont été ajoutées, **MAIL** et **RSV4**, permettant respectivement de lire un maillage au format GMSH ou CESAR (_mail.resu) et d'extraire les résultats disponibles dans un fichier .rsv4.

3.30.1 Classe MAIL

La classe **MAIL** permet de lire un maillage au format GMSH ou CESAR (fichier _mail.resu).
Le maillage réalisé avec GMSH doit satisfaire les conditions suivantes :

- le domaine doit être partitionné (au sens mathématique du terme, en particulier pas de recouvrements) en Physical (langage GMSH) en correspondance avec la partition souhaitée du domaine en groupes CESAR,
- aucun autre Physical ne doit être créé (en particulier pas de Physical Line en 2D ou de Physical Surface/Line en 3D) sinon GMSH introduit dans le fichier de maillage (.msh) des mailles non prévues (comme par exemple des mailles linéiques de bord en 2D, ou bien des mailles doubles, ...),
- les Physical doivent être numérotés de façon continue à partir de 1.

```
mail = MAIL(
        ♦ FIC = fic, [str]
        ♦ FMT = / 'CESAR',
                / 'GMSH',
                  ♦ NDIM = / 2,
                           / 3,
                  ♦ FAMI = fami, [dict {int : | 'MB',
                                               | 'MT',
                                               | 'PB',
                                               | 'PT',
                                               | 'BB',
                                               | 'BT',
                                               | 'CO',
                                               | 'AX',
                                               | 'DB',
                                               | 'DT',
                                               | 'EB',
                                               | 'ET',
                                               | 'SB',
                                               | 'OB',
                                               | 'OT'
                                       }] # fin dict
        ◊ FD = fd, [list <GFD>]
        ) # fin MAIL
```

FIC : chaîne de caractères donnant le nom du fichier de maillage à lire.
FMT : chaîne de caractères donnant le format du fichier de maillage, à choisir parmi **CESAR** et **GMSH**.
NDIM : entier donnant la dimension du maillage, à choisir parmi **2** et **3**.
FAMI : dictionnaire dont les clés sont des entiers et les valeurs des chaînes de caractères à choisir parmi **'MB'**, **'MT'**, **'PB'**, **'PT'**, **'BB'**, **'BT'**, **'CO'**,**'AX'**,**'DB'**, **'DT'**, **'EB'**, **'ET'**, **'SB'**, **'OB'**, **'OT'**. Cette donnée permet de définir les groupes d'éléments au sens de CESAR :
— les clés du dictionnaire correspondent aux numéros des groupes et doivent être numérotées de façon continue à partir de 1. Le fichier de maillage doit contenir les groupes de mailles (Physical en langage GMSH) correspondant et ainsi numérotés.
— les valeurs du dictionnaire correspondent aux familles au sens de CESAR (**'MB'** pour Mécanique Bidimensionnelle, ...) affectées aux groupes.
FD : liste d'objets de type **GFD** permettant de définir des groupes d'éléments de contact au sens de CESAR.

Description des groupes d'éléments de contact. La classe **GFD** permet de définir un groupe d'éléments de contact au sens de CESAR. Ces éléments, de forme non standard (quadrangles à 6 noeuds FDQ6 ou prisme à 12 noeuds FDP12 ou encore hexaèdre à 16 noeuds FDH16) ne sont pas disponibles dans les mailleurs usuels (notamment GMSH) et doivent donc être définis après lecture du fichier de maillage.

gfd = **GFD**(
♦ **NF** = nf, [**int**]
♦ **NMF** = / **3**,
/ **6**,
/ **8**,
♦ **NFP1** = nfp1, [**list**<**int**>]
♦ **NFP2** = nfp2, [**list**<**int**>]
♦ **KG** = kg, [**int**]
) # *fin GFD*

NF : entier donnant le nombre de facettes sur chaque bord du contact.
NMF : entier donnant le nombre maximal de noeuds par facette, à choisir parmi **3**, **6** ou **8**.
NFP1 : liste d'entiers donnant les numéros des noeuds des facettes du bord 1. La connectivité de chaque facette doit être conforme à la connectivité de référence de CESAR.
NFP2 : liste d'entiers donnant les numéros des noeuds des facettes du bord 2, en vis à vis des facettes du bord 1. La connectivité de chaque facette doit être conforme à la connectivité de référence de CESAR.
KG : entier donnant le numéro du groupe d'éléments (à choisir par continuité des numéros des groupes existants).

La connectivité de chaque élément de contact est alors déduite des données **NFP1** et **NFP2** de la façon suivante : les 2 (resp. 3, 4) premiers noeuds d'un élément FDQ6 (resp. FDP12, FDH16) correspondent aux premiers noeuds de la facette du bord 1.
On rappelle également qu'un élément de contact bidimensionnel (FDQ6) doit présenter une normale orientée vers les $Z > 0$. Il convient donc de définir les listes **NFP1** et **NFP2** de façon à ce que l'application de la règle précédente conduise à cette orientation.

3.30.2 Classe RSV4

La classe **RSV4** permet de récupérer les résultats d'un calcul CESAR en tant qu'objet Python à partir de la donnée d'un fichier de résultats CESAR (fichier .rsv4). L'objet ainsi instancié peut être exploré de façon à récupérer certaines valeurs qui peuvent alors faire l'objet d'un post-traitement.
Cette possibilité d'exploration reste une fonctionnalité avancée qui nécessite d'être familier avec la structure et les concepts associés (*entités*, *tables*, *datasets*) d'un fichier .rsv4. Cette structure est décrite dans le document "Format des fichiers de résultats .rsv4" de la documentation de CESAR.
Par exemple, à la fin d'un script en langage CESAR, le codage suivant :

```
jeu.lancer()

rsv4 = RSV4(jeu, 'ssls_M011.rsv4')

comp = ['SXXU','SYYU','SZZU','SXYU','SYZU','SZXU',
        'SXXL','SYYL','SZZL','SXYL','SYZL','SZXL',
        'NXX','NYY','NZZ','NXY','NYZ','NZX',
        'MXX','MYY','MZZ','MXY','MYZ','MZX']
for entity in rsv4.entities:
   if isinstance(entity, LoadingCase):
      for table in entity.tables:
         if isinstance(table, ElementsByGroup) and table.type_dataset == 'Shell Stress tensor':
            myy = table.val[numa][6][comp.index('MYY')]
```

- lance le calcul CESAR,
- fait remonter les résultats du calcul en Python (instanciation de l'objet rsv4) par lecture du fichier de résultats `ssls_M011.rsv4`,
- explore l'objet rsv4 à la recherche, pour chaque *entité* de type **LoadingCase**, de la valeur de la composante 'MYY' du *dataset* de type **Shell Stress tensor** de la *table* de type **ElementsByGroup**, au noeud d'index 6 de la maille de numéro numa.

Il est également possible de formater "en clair" les résultats stockés dans le fichier binaire .rsv4, au moyen de la méthode **formater** de la classe **RSV4**.
Par exemple, le codage suivant :

```
rsv4.formater(jeu, 'ssls_M011.frsv4')
```

formate les résultats du calcul dans le fichier texte `ssls_M011.frsv4`.
Enfin, on peut exporter les résultats au format GMSH via la méthode **exporter** de la classe **RSV4** :

```
rsv4.exporter(jeu, 'GMSH', 'ssls_M011.pos')
```

où `ssls_M011.pos` est un fichier au format GMSH contenant le maillage ainsi que les résultats.

3.31 Exemple de jeu de données

A titre d'exemple, on reproduit ci-dessous le script correspondant au cas test ssnp007, qui illustre un calcul de contact pour un modèle plan.
On commente ensuite les différentes parties de la mise en données.

3.31.1 Script

```
1 from modele_cesar import *
2
3 rep_trav = 'tests/atelier/'
4 fic_data = rep_trav + 'ssnp_M007.data'
```

```
fic_rsv4 = rep_trav + 'ssnp_M007.rsv4'
fic_msh = rep_trav + 'ssnp_M007.msh'


exe = EXEC(DATA = fic_data,
           RSV4 = fic_rsv4,
           MSH = fic_msh)

comt = COMT(LIGNES = ["-"*72, "exemple :","contact en 2D", "-"*72])

expo = EXPO(IPGMSH = 1)

coor = COOR(M = 2,
            M1 = 0,
            NNT = 56,
            NDIM = 2,
            VCORG = [ 0.,   0.,
                      1.,   0.,
                      0.,   0.5,
                      0.,   1.,
                      1.,   1.,
                      0.,   1.,
                      1.,   2.,
                      0.,   1.5,
                      1.,   1.,
                      2.,   0.,
                      3.,   0.,
                      2.,   0.5,
                      2.,   1.,
                      3.,   1.,
                      0.,   2.,
                      2.,   1.,
                      3.,   2.,
                      2.,   1.5,
                      3.,   1.,
                      2.,   2.,
                      4.,   0.,
                      5.,   0.,
                      4.,   0.5,
                      4.,   1.,
                      5.,   1.,
                      4.,   1.,
                      5.,   2.,
                      4.,   1.5,
                      5.,   1.,
                      4.,   2.,
                      6.,   0.,
                      7.,   0.,
                      6.,   0.5,
                      6.,   1.,
                      7.,   1.,
                      6.,   1.,
                      7.,   2.,
                      6.,   1.5,
                      7.,   1.,
                      6.,   2.,
                      8.,   0.,
                      9.,   0.,
                      10.,  0.5,
                      8.,   0.5,
                      10.,  0.,
                      8.,   1.,
                      9.,   1.,
                      8.,   1.,
                      9.,   2.,
                      8.,   1.5,
                      9.,   1.,
                      10.,  1.5,
                      8.,   2.,
                      10.,  1.,
                      10.,  1.,
                      10.,  2.  ])

elem = ELEM(M = 2,
            M1 = 0,
```

```
80              NELT = 15,
81              NGRPE = 3,
82              PNUMEL = [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 87, 93, 99, 105, 111],
83              NUMEL = [ 1, 10, 13,  4,  2, 12, 5,  3,
84                        10, 21, 24, 13, 11, 23, 14, 12,
85                        21, 31, 34, 24, 22, 33, 25, 23,
86                        31, 41, 46, 34, 32, 44, 35, 33,
87                        41, 45, 54, 46, 42, 43, 47, 44,
88                        56, 53, 48, 55, 49, 50, 51, 52,
89                        53, 40, 36, 48, 37, 38, 39, 50,
90                        40, 30, 26, 36, 27, 28, 29, 38,
91                        30, 20, 16, 26, 17, 18, 19, 28,
92                        20, 15,  6, 16,  7,  8,  9, 18,
93                        55, 48, 46, 54, 51, 47,
94                        48, 36, 34, 46, 39, 35,
95                        36, 26, 24, 34, 29, 25,
96                        26, 16, 13, 24, 19, 14,
97                        16,  6,  4, 13,  9,  5 ],
98              TYPE = ['MBQ8', 'MBQ8', 'MBQ8', 'MBQ8', 'MBQ8',
99                      'MBQ8', 'MBQ8', 'MBQ8', 'MBQ8', 'MBQ8',
100                     'FDQ6', 'FDQ6', 'FDQ6', 'FDQ6', 'FDQ6'],
101             GROUPE = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3],
102             GRPES = [MB(NOMG = 'G1',
103                         ACTI = 'A',
104                         IMOD = 1,
105                         CARA = MB_ELI(RO = 0.02,
106                                       YOUNG = 0.1e4,
107                                       POISS = 0.2,
108                                       INAT = 1)),
109                      MB(NOMG = 'G2',
110                         ACTI = 'A',
111                         IMOD = 1,
112                         CARA = MB_ELI(RO = 0.02,
113                                       YOUNG = 0.1e4,
114                                       POISS = 0.2,
115                                       INAT = 1)),
116                      FD(NOMG = 'G3',
117                         ACTI = 'A',
118                         IMOD = 2,
119                         ITAC = 1,
120                         CARA = FD_FC(COERI = 0.1e4,
121                                      RT = 0.,
122                                      C = 0.,
123                                      PHI = 30.,
124                                      PSI = 30.,
125                                      INAT = 1)) ])
126
127 cond = COND(M = 2,
128             OPT = [COND_NUL(GEN = [GEN_NUL(IGEN = 2,
129                                            NP = 6,
130                                            NUM = [4, 3, 1, 15, 8, 6],
131                                            IDL = [1, 0],
132                                            NDLN = 2),
133                                    GEN_NUL(IGEN = 2,
134                                            NP = 6,
135                                            NUM = [4, 3, 1, 15, 8, 6],
136                                            IDL = [0, 1],
137                                            NDLN = 2)])])
138
139 char = CHAR(M = 4,
140             OPT = [CHAR_POI()])
141
142 char2 = CHAR(M = 4,
143              OPT = [CHAR_PUR(NF = 5,
144                              NMF = 3,
145                              NFP = [56, 53, 49,
146                                     53, 40, 37,
147                                     40, 30, 27,
148                                     30, 20, 17,
149                                     20, 15, 7],
150                              P = -1)])
151
152 tcnl = TCNL(M = 2,
153             NINCR = 2,
154             NITER = 50,
```

```
155                 TOL = 0.00001,
156                 IAUTO = 0,
157                 KNP = 1,
158                 KND = 1,
159                 KNF = 1,
160                 VFT = [1., 1., -0.1, 0.05])
161
162 jeu = JDD(EXEC = exe,
163            COMT = comt,
164            EXPO = expo,
165            COOR = coor,
166            ELEM = elem,
167            COND = cond,
168            CHAR = [char, char2],
169            TCNL = tcnl)
170
171 jeu.lancer()
```

3.31.2 Commentaires

Analysons les différentes parties du script précédent.

Préambule. On commence par importer le langage CESAR :

```
from modele_cesar import *
```

puis on définit les noms des fichiers à gérer à l'aide des variables `fic_data`, `fic_rsv4` et `fic_msh`, en utilisant la variable `rep_trav` pour stocker le chemin d'accès commun à ces fichiers :

```
rep_trav = 'tests/atelier/'
fic_data = rep_trav + 'ssnp_M007.data'
fic_rsv4 = rep_trav + 'ssnp_M007.rsv4'
fic_msh = rep_trav + 'ssnp_M007.msh'
```

Données utilitaires. On commence par définir les caractéristiques d'un passage CESAR en spécifiant le mode exécution et en définissant les fichiers en entrée (.data) et en sortie du calcul (.rsv4 pour le format natif de CESAR et .msh pour le format GMSH), en utilisant les variables `fic_data`, `fic_rsv4` et `fic_msh` définies précédemment. On choisit de nommer `exe` l'instance de **EXEC** :

```
exe = EXEC(DATA = fic_data,
           RSV4 = fic_rsv4,
           MSH = fic_msh)
```

puis on définit quelques lignes de commentaires, en choisissant de nommer `comt` l'instance de **COMT** :

```
comt = COMT(LIGNES = ["-"*72, "exemple :","contact en 2D", "-"*72])
```

et enfin on demande l'exportation au format GMSH du maillage et des résultats du calcul :

```
expo = EXPO(IPGMSH = 1)
```

Données de modèle. On définit ensuite l'ensembles des données relatives :

— aux noeuds :

```
coor = COOR(M = 2,
            M1 = 0,
            NNT = 56,
            NDIM = 2,
            VCORG = [ 0.,   0.,
                      1.,   0.,
                      ...
```

```
                        10.,   1.,
                        10.,   2.   ])
```

On définit 56 noeuds en dimension 2 dont les coordonnées sont données dans une liste affectée à **VCORG**.
On choisit de nommer `coor` l'instance de **COOR**.

— aux éléments :

```
elem = ELEM(M = 2,
            M1 = 0,
            NELT = 15,
            NGRPE = 3,
            PNUMEL = [1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 87, 93, 99, 105, 111],
            NUMEL = [ 1, 10, 13,  4,  2, 12, 5,  3,
                      10, 21, 24, 13, 11, 23, 14, 12,
                      ...
                      26, 16, 13, 24, 19, 14,
                      16,  6,  4, 13,  9,  5 ],
            TYPE = ['MBQ8', 'MBQ8', 'MBQ8', 'MBQ8', 'MBQ8',
                    'MBQ8', 'MBQ8', 'MBQ8', 'MBQ8', 'MBQ8',
                    'FDQ6', 'FDQ6', 'FDQ6', 'FDQ6', 'FDQ6'],
            GROUPE = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3],
            GRPES = [MB(NOMG = 'G1',
                        ACTI = 'A',
                        IMOD = 1,
                        CARA = MB_ELI(RO = 0.02,
                                      YOUNG = 0.1e4,
                                      POISS = 0.2,
                                      INAT = 1)),
                     MB(NOMG = 'G2',
                        ACTI = 'A',
                        IMOD = 1,
                        CARA = MB_ELI(RO = 0.02,
                                      YOUNG = 0.1e4,
                                      POISS = 0.2,
                                      INAT = 1)),
                     FD(NOMG = 'G3',
                        ACTI = 'A',
                        IMOD = 2,
                        ITAC = 1,
                        CARA = FD_FC(COERI = 0.1e4,
                                     RT = 0.,
                                     C = 0.,
                                     PHI = 30.,
                                     PSI = 30.,
                                     INAT = 1)) ])
```

On définit 15 éléments répartis en 3 groupes. Leurs connectivités nodales sont définies dans une liste affectée à **NUMEL** dans laquelle on se repère grâce à la liste des pointeurs sur le début des éléments affectée à **PNUMEL**. Les types des éléments figurent dans la liste affectée à **TYPE** et les appartenances des éléments aux groupes sont données par la liste affectée à **GROUPE**. La donnée des caractéristiques des groupes d'éléments est introduite via le mot-clé **GRPES** qui reçoit une liste de 3 blocs :

- 1 bloc pour un groupe nommé `'G1'` d'éléments de mécanique bidimensionnelle affectés d'une loi de type élasticité linéaire,
- 1 bloc pour un groupe nommé `'G2'` d'éléments de mécanique bidimensionnelle affectés d'une loi de type élasticité linéaire,
- 1 bloc pour un groupe nommé `'G3'` d'éléments de contact affectés d'une loi de type frottement de Coulomb.

On choisit de nommer `elem` l'instance de **ELEM**.

— aux conditions limites :

```
cond = COND(M = 2,
            OPT = [COND_NUL(GEN = [GEN_NUL(IGEN = 2,
                                           NP = 6,
                                           NUM = [4, 3, 1, 15, 8, 6],
```

```
                                                IDL = [1, 0],
                                                NDLN = 2),
                                        GEN_NUL(IGEN = 2,
                                                NP = 6,
                                                NUM = [4, 3, 1, 15, 8, 6],
                                                IDL = [0, 1],
                                                NDLN = 2)])])
```

La donnée des conditions aux limites est introduite par le mot-clé **OPT** qui reçoit une liste de 2 options :

- 1 option de type ddl imposés nuls concernant le premier des 2 ddl de chacun des 6 noeuds dont les numéros figurent dans la liste affectée à **NUM**,
- 1 option de type ddl imposés nuls concernant le second des 2 ddl de chacun des 6 noeuds dont les numéros figurent dans la liste affectée à **NUM**.

On choisit de nommer `cond` l'instance de **COND**.

— aux chargements :

La donnée d'un premier chargement est introduite par le mot-clé **OPT** qui reçoit une liste de 1 option de type poids propre.

On choisit de nommer `char` cette première instance de **CHAR**.

```
char = CHAR(M = 4,
            OPT = [CHAR_POI()])
```

On introduit ensuite un deuxième chargement défini par une option de type pression uniformément répartie de valeur -1 et appliquée sur les 5 facettes d'éléments définies par la liste des numéros de noeuds affectée à **NFP**.

On choisit de nommer `char2` cette deuxième instance de **CHAR**.

```
char2 = CHAR(M = 4,
             OPT = [CHAR_PUR(NF = 5,
                             NMF = 3,
                             NFP = [56, 53, 49,
                                    53, 40, 37,
                                    40, 30, 27,
                                    30, 20, 17,
                                    20, 15, 7],
                             P = -1)])
```

Données de calcul. Il reste à définir les données du problème de contact à résoudre. On choisit de pilotes ce calcul en 2 incréments de chargements en autorisant au plus 50 itérations par incrément avec une tolérance de $1.e-5$ sur la convergence. Les coefficients multiplicatifs à appliquer aux 2 chargements (`char` et `char2`) pour les 2 incréments de calcul sont donnés par la liste affectée à **VFT**. Les critères de non pénétration, de décollement et de frottement doivent être vérifiés.

On choisit de nommer `tcnl` cette deuxième instance de **TCNL**.

```
tcnl = TCNL(M = 2,
            NINCR = 2,
            NITER = 50,
            TOL = 0.00001,
            IAUTO = 0,
            KNP = 1,
            KND = 1,
            KNF = 1,
            VFT = [1., 1., -0.1, 0.05])
```

Jeu de données. On termine le script en définissant le jeu de données à exécuter en faisant référence à l'ensemble des données :

— utilitaires (`exe` et `comt`),

— de modèle (`coor`, `elem`, `cond`, `char` et `char2`),
— de calcul (`tcnl`),

et en définissant ses paramètres d'exécution (ici on distribue les 2 résolutions sur 2 processeurs). On choisit de nommer `jeu` l'instance de **JDD** :

```
jeu = JDD(EXEC = exe,
          COMT = comt,
          EXPO = expo,
          COOR = coor,
          ELEM = elem,
          COND = cond,
          CHAR = [char, char2],
          TCNL = tcnl)
```

Il reste à demander l'exécution du jeu de données via sa méthode **lancer** :

```
jeu.lancer()
```

Chapitre 4

Installation

4.1 Généralités

Le Pilote de CESAR utilise les librairies open source **xdata** et **MED Mémoire** développées par le CEA dans le cadre du développement de sa plate-forme SALOME.

xdata. La librairie xdata est un module Python utilisé par le Pilote pour construire ses modèles de données :

- son modèle de données "pivot", correspondant au langage de commandes Pilote,
- son modèle de données CESAR, correspondant au langage de commandes CESAR.

Ce module permet de définir statiquement les types des données attendues par le Pilote de façon à pouvoir vérifier les données rentrées par l'utilisateur en amont de l'exécution, ce qui n'est pas possible en Python natif puisque c'est un langage à typage dynamique.

MED Mémoire. La librairie MED Mémoire est une librairie C++ permettant de stocker en mémoire les données relatives aux maillages et aux champs de résultats. Cette librairie est dotée d'une API Python que le Pilote utilise pour les attributs **mesh** de la classe **MAILLAGE** et **fields** de la classe **RESULTAT**. MED Mémoire s'appuie sur la librairie **MED Fichier** (développée par EDF) qui dépend elle de la librairie **HDF5**.

L'installation du Pilote doit donc commencer par l'installation de ces 4 prérequis : xdata, HDF5, MED Fichier, MED Mémoire.

4.2 Linux

L'installation a été testée sous Debian-6, Debian-7, Ubuntu-12.04, Ubuntu-14.04.

Paquets de base. Pour une installation nouvelle du système, on peut commencer par l'installer à partir d'une image iso (test réalisé avec l'image `debian-6.0.4-amd64-netinst.iso`), puis installer les paquets de base suivants : `make`, `gcc`, `swig`, `g++`, `gfortran`, `tk`, `zlib1g-dev`, `automake`, `libtool`, `libboost-dev`, `libboost-dbg`, `libboost-thread-dev`, `python-dev`, `libxml2-dev`, `doxygen`, `python-qt4`, `python-vtk`, `nedit`, `gmsh`.
Pour un système déjà installé et utilisé pour du calcul scientifique, la plupart de ces paquets sont en général déjà présents.

Code_Aster. Si l'on souhaite utiliser le Pilote pour lancer des calculs avec le solveur Code_Aster (version 11.1), il convient de réaliser son installation au préalable. Une archive de Code_Aster (version 11.1) est fournie dans l'archive du Pilote. L'installation de Code_Aster doit être effectuée en suivant les instructions fournies dans le fichier `README_aster.txt`.

Pilote de CESAR. Après avoir décompressé l'archive, les instructions d'installation sont fournies dans le fichier `README.txt`. Les étapes à suivre sont les suivantes :

- éditer le fichier de configuration `config.txt` de façon à fixer certaines variables d'environnement :
  - `REP_INSTAL` définissant le répertoire d'installation du Pilote,
  - `REP_ASTER` définissant le répertoire d'installation de Code_Aster (le cas échéant),
  - `REP_TMP` définissant le répertoire temporaire d'exécution des calculs,
  - `VID_TMP` définissant le choix du vidage systématique ou non du répertoire temporaire d'exécution des calculs,
- lancer l'installation du Pilote en exécutant le script bash `install.sh`.

4.3 Windows

Bien que le Pilote soit plutôt dédié aux utilisateurs Linux, une version Windows 32 bits est fournie. Celle-ci contient les différents outils (GMSH et SALOME) et prérequis (Python, PyQT, HDF5, MED Fichier, MED Mémoire, XDATA, ...) déjà installés. Il suffit donc de décompresser l'archive fournie puis d'ajuster certaines variables en fonction du choix du répertoire d'installation (lire pour cela le fichier `README.txt` fourni à la racine de la distribution).
Parmi les outils, seul manque le solveur Code_Aster, EDF n'en distribuant pas de version Windows. Aucun appel à Code_Aster ne peut donc être réalisé. Néanmoins, la fonctionnalité d'export d'une étude au format Code_Aster reste disponible.

4.4 Test de l'installation

Script de test. Une fois l'installation du Pilote terminée, on peut la tester en appelant le script `run_test` (présent dans le répertoire `bin/` de la distribution du Pilote) qui lance l'exécution d'un test parmi l'ensemble de ceux présents dans le répertoire `pilote/tests/donnees`. Le choix du test se fait en éditant le script `run_test` et en modifiant la variable `FICHIERS`. Par défaut, celle-ci est fixée ainsi :

```
FICHIERS="sslp001e.py sslp001e.msh" #API Pilote, calcul CESAR
```

ce qui signifie que le test par défaut est le test de nom `sslp001e` défini par le script `sslp001e.py` (utilisant le langage de commandes Pilote pour lancer un calcul CESAR) et par le fichier de maillage `sslp001e.msh` (format GMSH).

Coloration syntaxique avec NEdit. Pour éditer un script Python utilisant le langage Pilote ou le langage CESAR, on peut utiliser l'éditeur NEdit muni du fichier de configuration `nedit.rc_PILOTE` fourni dans l'archive du Pilote. Ce fichier est alors à placer à la racine du répertoire caché `.nedit` en le renommant `nedit.rc` (remplaçant alors le `nedit.rc` original). Il permet, au lancement de NEdit, de choisir le langage `Pilote` (dans `Preferences` / `Langage Mode`) qui fournit une coloration syntaxique adaptée aux langages Pilote et CESAR. Il permet également de disposer d'une commentarisation / décommentarisation automatique de blocs de textes (via `Macro` / `Comments` / `# Comment` ou `#Uncomment`).

Chapitre 5

Bibliothèque d'exemples

5.1 Présentation générale

Cette section présente sous la forme de vues synoptiques l'ensemble des tests utilisés pour vérifier la non régression du Pilote au cours de son développement. Du point de vue de l'utilisateur, ces tests peuvent être vus comme autant d'exemples d'utilisation du Pilote et il paraît donc intéressant de fournir à l'utilisateur un panorama de l'ensemble des tests disponibles.
Pour chaque test, on donne l'allure du maillage utilisé ainsi qu'une liste de mots-clé (noms de classes ou littéraux d'énumération du langage utilisé) permettant de caractériser le test.
On rappelle également que pour connaître l'ensemble des tests utilisant un mot-clé donné, il suffit d'utiliser l'utilitaire `GREP` dans un terminal. Par exemple, pour le mot-clé **STAT_NON_LINE**, après s'être déplacé dans le répertoire `pilote/tests/donnees` du répertoire d'installation du Pilote, on tape la commande :

```
grep -n 'STAT_NON_LINE' *.py
```

Classification. Les tests ont été nommés en s'inspirant de la codification proposée par le "Guide de validation des progiciels de calcul de structures" édité par l'AFNOR (1990). Cette codification est également employée pour le nommage des cas tests de validation du solveur Code_Aster. Le nom d'un test est constitué :

- d'une série de 4 caractères alphabétiques définissant sa **catégorie** en indiquant successivement :
  - le domaine d'application (**s** pour structure, **t** pour thermique, **h** pour hydrogéologie, **p** pour milieu poreux),
  - le type d'analyse (**s** pour statique, **d** pour dynamique, **p** pour permanent, **t** pour transitoire),
  - le type de comportement (**l** pour linéaire, **n** pour non linéaire),
  - le type de modèle (**l** pour linéique, **p** pour plan, **v** pour volumique, **s** pour surfacique).
- suivie d'un numéro à 3 chiffres indiquant le **numéro** du test dans la catégorie,
- suivi éventuellement d'un caractère alphabétique indiquant la **variante** du test.

La catégorie **zzzz** est réservée à la validation de fonctionnalités informatiques spécifiques.

Tests du langage Pilote. Les tests suivants utilisent le langage Pilote :

- catégorie **ssll** : 001a+b, 002a, 003a+b, 004a, 005a+b, 006a, 008a, 009a
- catégorie **sslp** : 001a, 001b+c+d+e+f+g+h+i+j+k+l+m, 002a+b+c+d+e+f, 004a, 005a+b, 006a+b, 007a, 008a, 009a, 011a+b+c, 014, 015, 016
- catégorie **ssls** : 002a+b+c+d, 003a+b+c+d, 004a+b, 005a+b, 007a, 008a, 010a+b

— catégorie **sslv** : 001a+b+c+d+e+f+g+h, 002a+b, 003a, 004a, 005a, 006a, 007a, 008a+b+c+d, 009a
— catégorie **ssnp** : 001a, 002a, 003a+b, 004a, 005a, 006a+b, 014a+b+c+d+e+f+g, 015a+b, 024a+b, 025, 026, 028, 029, 030, 032, 033, 035
— catégorie **ssnv** : 001a, 002a+b, 007
— catégorie **sdls** : 001a+b+c+d, 002a+b+c+d+e, 003a+b, 004a+b
— catégorie **sdlv** : 001a, 002a
— catégorie **stnp** : 002
— catégorie **stnv** : 002
— catégorie **zzzz** : 001a+b+c, 002, 003, 004a+b+c+d, 005a+b, 006, 007, 008, 009

Dans la liste précédente :

— les tests zzzz001a+b+c et sslp016 constituent des exemples où le maillage est défini "manuellement" à l'aide de l'**API Python de MED Mémoire** incluse dans le langage Pilote
— les tests zzzz002, zzzz003, zzzz004a+b+c+d, zzzz005a+b constituent des exemples de **conversion** de maillage
— les tests ssnp030, stnp002, zzzz006, zzzz007, zzzz008, zzzz009 constituent des exemples de définition géométrique paramétrée du domaine
— les tests :
  · sdls : 002a+b+c+d+e
  · sdlv : 002a
  · ssll : 002a, 004a, 006a, 009a
  · sslp : 004a, 006a+b, 008a, 011c
  · ssls : 003a+b+c+d, 005a+b, 008a
  · sslv : 002a+b, 004a, 006a, 008c
  · ssnp : 002a, 014d+e+f

  constituent des exemples d'appel du solveur **Code_Aster**.

Tests du langage CESAR. Les tests suivants utilisent le langage CESAR :

— catégorie **ssll** : 007, 010a+b
— catégorie **sslp** : 003, 010, 012, 013
— catégorie **ssls** : 001, 006, 009a+b+c, 011
— catégorie **ssnl** : 001a+b+c+d+e
— catégorie **ssnp** : 007, 008a+b+c+d, 009a+b, 010a+b, 011a+b, 012, 013, 016a+b, 017, 018, 019, 020, 021, 022, 023, 027a+b, 031, O34, 035
— catégorie **ssns** : 001a+b+c+d+e
— catégorie **ssnv** : 003, 004a+b+c+d, 005a+b, 006
— catégorie **sdll** : 001a+b
— catégorie **sdlv** : 001a, 002a
— catégorie **stnp** : 001
— catégorie **stnv** : 001
— catégorie **tplp** : 001
— catégorie **ttlp** : 001
— catégorie **ttnp** : 001, 002
— catégorie **ttnv** : 001, 002
— catégorie **ptlp** : 001
— catégorie **ptlv** : 001
— catégorie **ptnp** : 001
— catégorie **ptnv** : 001

— catégorie **hpnp** : 001
— catégorie **htnp** : 001, 002

Tests avec scripting. Les tests suivants exploitent des éléments du langage Python (boucles, tests, fonctions, ...) et constituent des exemples de possibilités de scripting :

— catégorie **sdls** : 001c+d, 002c+d, 003a+b, 004a+b
— catégorie **sslp** : 001k, 010
— catégorie **ssls** : 011
— catégorie **ssnp** : 003a, 004a, 005a, 006a, 014g, 020, 030, 033
— catégorie **ssnv** : 001a, 002a

Dans la liste précédente :

— les tests sslp001k, ssnp029 et ssnp030 constituent des exemples d'**étude paramétrée**,
— le test sslp030 montre un exemple d'extraction de grandeurs (champ de coordonnées, champ de résultats),
— le test sslp010 montre un exemple simple d'utilisation de Python pour définir les noeuds et les éléments d'un maillage plan structuré ainsi que les zones d'application des charges et conditions limites,
— le test ssls011 montre un exemple de calcul de surface d'influence (flexion locale d'un hourdis) illustrant l'utilisation de la classe **RSV4** pour la récupération et l'exploitation en Python de résultats de calcul,
— le test sdls003a+b montre un exemple d'utilisation de l'**API Python de Salomé** pour construire la géométrie (module **GEOM**) et le maillage (module **SMESH**),
— le test sdls004a+b montre :
  . une utilisation avancée de l'**API Python de Salomé** (modules **GEOM** et **SMESH**) pour la construction de la géométrie et du maillage,
  . une utilisation avancée de Python pour la mise en données et l'exploitation des résultats du calcul, à la fois par l'utilisation de l'**API Python de MED Mémoire** et par la programmation de fonctions utilisateur.

  Ce test reprend une étude réalisée par la société NECS avec Salomé et Code_Aster dans le cadre de SIPRIS, projet de recherche FUI auquel le laboratoire LISIS a participé.

Tests avec phasage. Les tests suivants mettent en oeuvre un phasage de construction :

— catégorie **ssnp** : 021, 022, 028, 029, 31, 32, 33

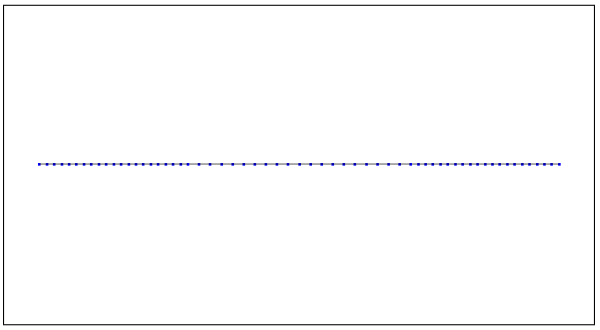
Dans la liste précédente :

— les tests ssnp021+022+028 traitent la construction phasée dun mur de soutènement (étude reprise du didacticiel "staged construction of a gravity wall" élaboré par la société ITECH pour l'interface CLEO) mais avec quelques différences :
  . le test ssnp021 utilise le langage CESAR et un fichier de maillage au format GMSH,
  . le test ssnp022 utilise le langage CESAR et un fichier de maillage au format CESAR (_mail.resu),
  . le test ssnp028 utilise le langage Pilote et un fichier de maillage au format GMSH.
— le test ssnp029 traite la construction phasée d'un tunnel (étude reprise du didacticiel "staged construction of a tunnel" élaboré par la société ITECH pour l'interface CLEO) avec le langage Pilote.

5.2 Catégorie "structure statique linéaire linéique" (ssll)

5.2.1 Test ssll001a

Allure du maillage.

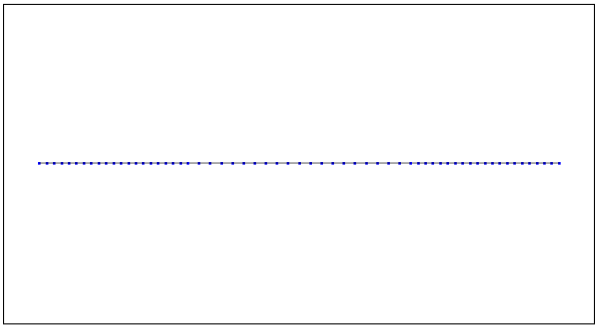


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'POUT_2D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **POUT_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_POUT_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.2.2 Test ssll001b

Allure du maillage.



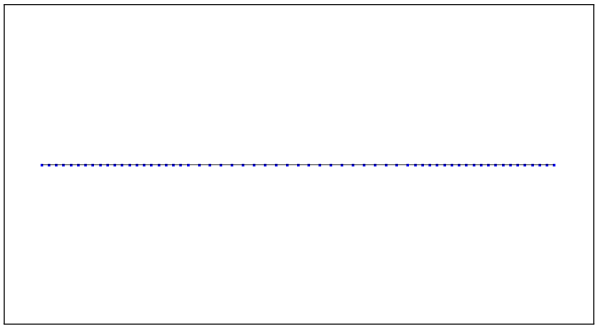
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'POUT_2D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **POUT_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_POUT_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FILTRE**, **FICHIER**, **'GMSH'**

— Etude : **lancer**

5.2.3 Test ssll002a

Allure du maillage.

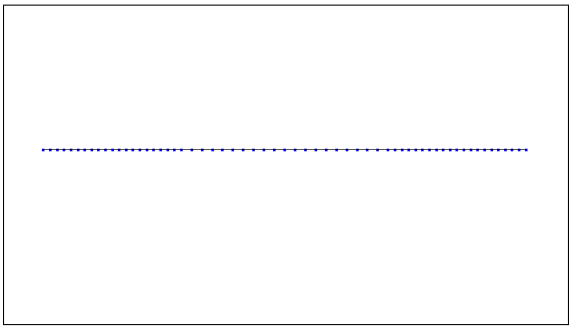


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'POUT_3D'**
— Matériaux : **ASTER_ELAS**
— Caractéristiques : **POUT_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_MAILLE_POUT_3D**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.2.4 Test ssll003a

Allure du maillage.



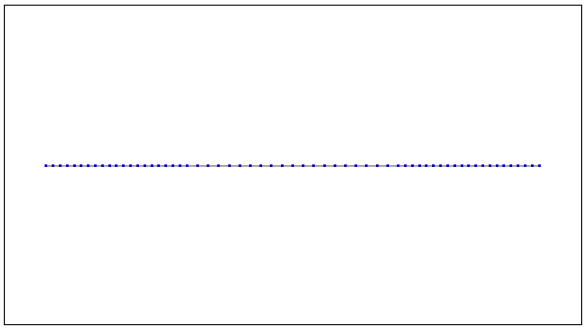
Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'POUT_2D'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques : **POUT_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_NOEUD**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**

— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.2.5 Test ssll003b

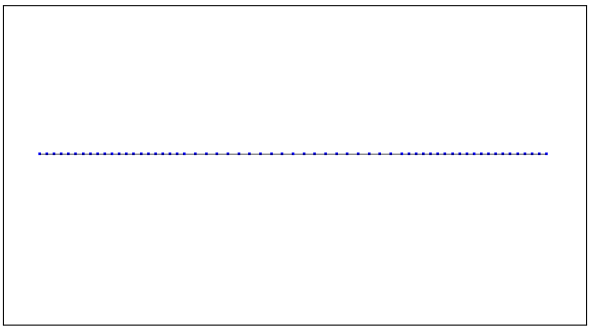
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'POUT_3D'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques : **POUT_3D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_NOEUD**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.2.6 Test ssll004a

Allure du maillage.

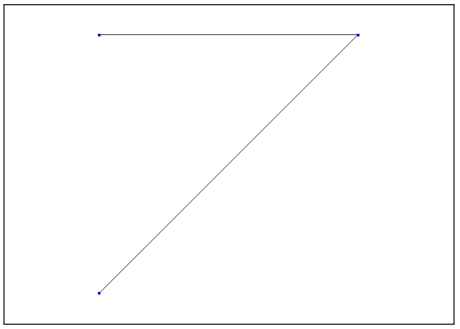


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'POUT_3D'**
— Matériaux : **ASTER_ELAS**
— Caractéristiques : **POUT_3D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_NOEUD**

— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.2.7 Test ssll005a

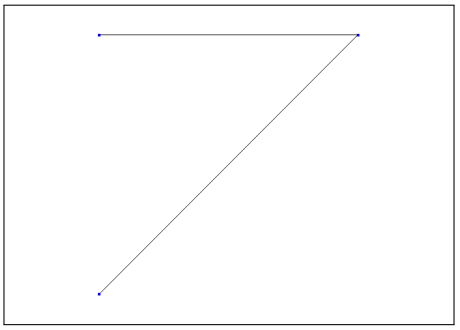
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'BAR_2D'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques : **BAR_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_NOEUD**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.2.8 Test ssll005b

Allure du maillage.

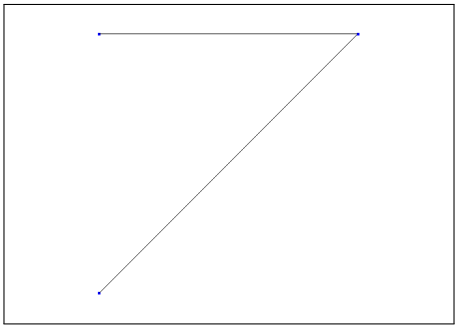


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'BAR_3D'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques : **BAR_3D**
— Conditions limites : **DDL_IMPOSE**

— Chargements : **FORC_NOEUD**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.2.9 Test ssll006a

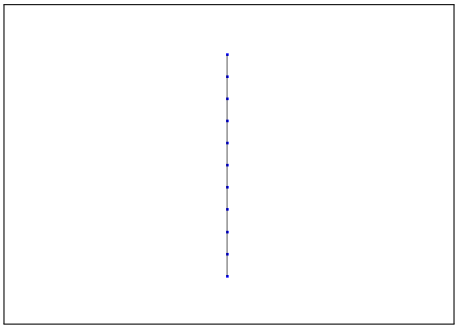
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'BAR_3D'**
— Matériaux : **ASTER_ELAS**
— Caractéristiques : **BAR_3D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_NOEUD**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.2.10 Test ssll007

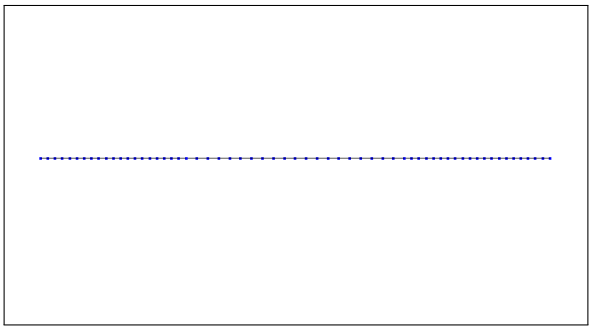
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **PT**, **PT_MF PT_LF**, **PT_LF_ELI**, **PT_CF**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_SOL**, **GEN_SOL**
— Calcul : **LINE**
— Jeu de données : **lancer**

5.2.11 Test ssll008a

Allure du maillage.

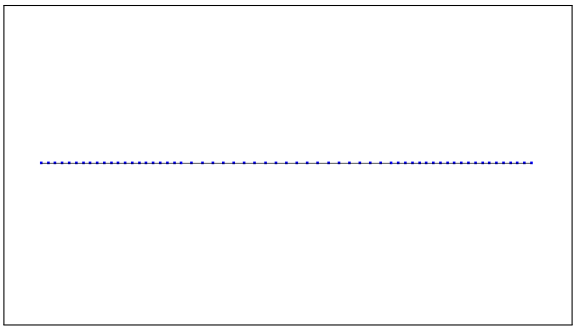


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'POUT_3D_MULTI'**
- Matériaux : **COMPOSITE**, **ELAS_LINE_ISO**
- Caractéristiques : **POUT_3D_MULTI**, **GROUP_FIBRE**, **FIBRE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_NOEUD**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.2.12 Test ssll009a

Allure du maillage.

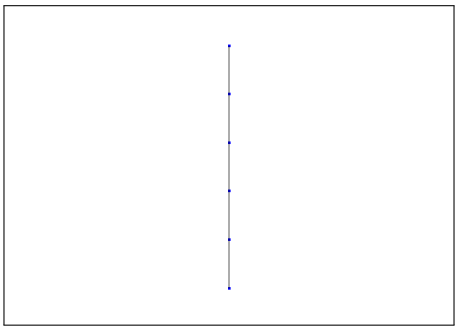


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'POUT_3D_MULTI'**
- Matériaux : **COMPOSITE**, **ASTER_ELAS**
- Caractéristiques : **POUT_3D_MULTI**, **GROUP_FIBRE**, **FIBRE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_NOEUD**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.2.13 Test ssll010a

Allure du maillage.

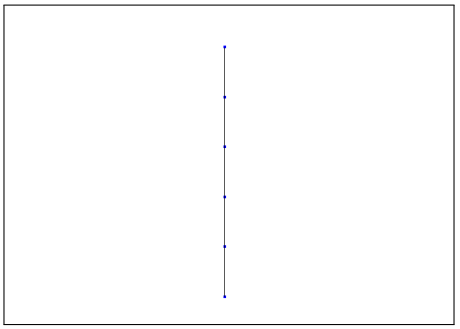


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **PB**, **PB_ELI SP**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.2.14 Test ssll010b

Allure du maillage.



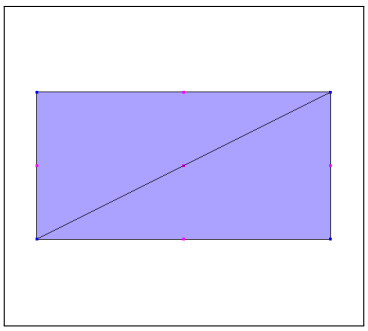
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **PT**, **PT_STD_ELI SP**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.3 Catégorie "structure statique linéaire plane" (sslp)

5.3.1 Test sslp001a

Allure du maillage.

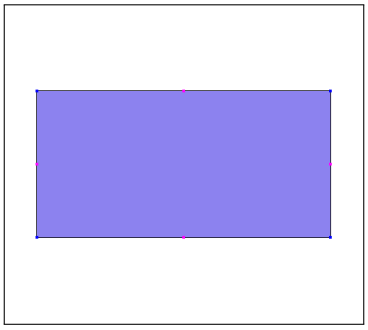


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.2 Test sslp001b

Allure du maillage.

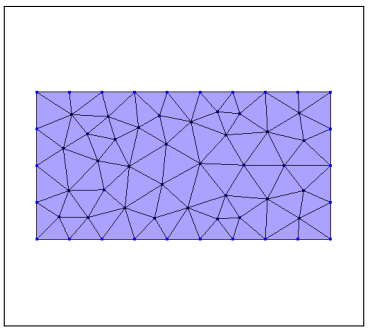


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.3 Test sslp001c

Allure du maillage.

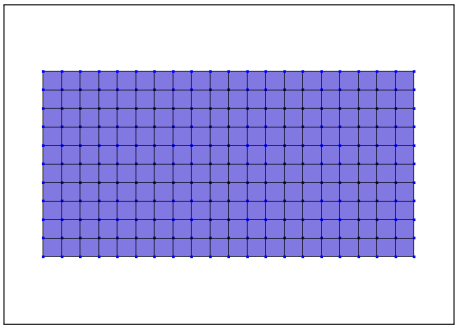


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.4 Test sslp001d

Allure du maillage.

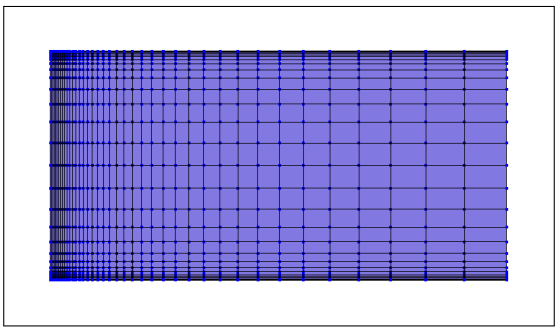


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.5 Test sslp001e

Allure du maillage.

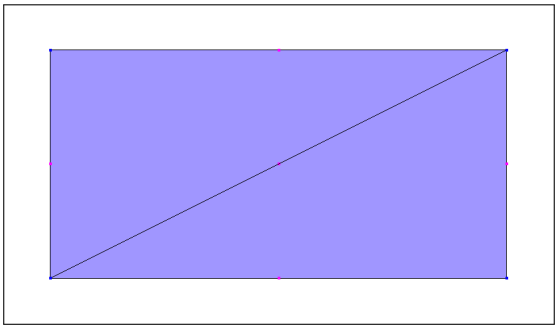


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.6 Test sslp001f

Allure du maillage.

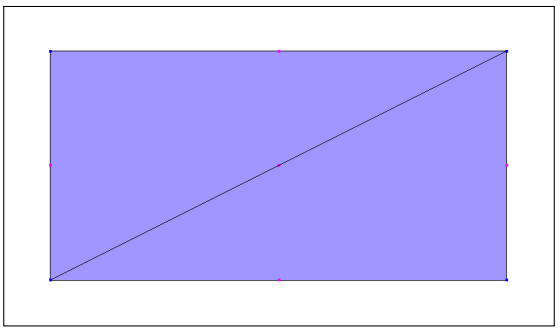


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.7 Test sslp001g

Allure du maillage.

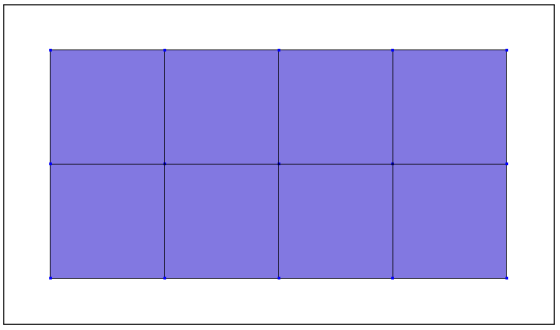


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.8 Test sslp001h

Allure du maillage.

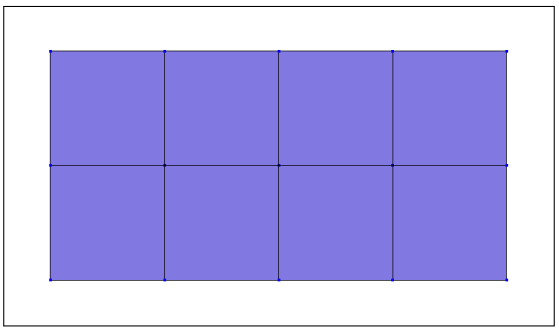


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.9 Test sslp001i

Allure du maillage.

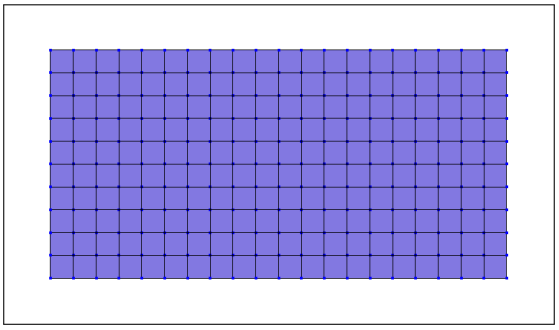


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.10 Test sslp001j

Allure du maillage.

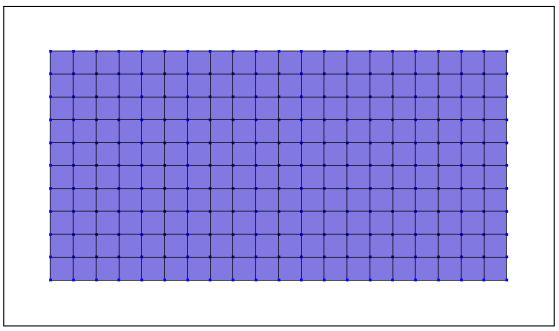


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**, **SELECT**, **'TXT'**, **DEPLA_2D**, **CONTR_2D**, **REAC_2D**
- Etude : **lancer**

5.3.11 Test sslp001k

Allure du maillage.

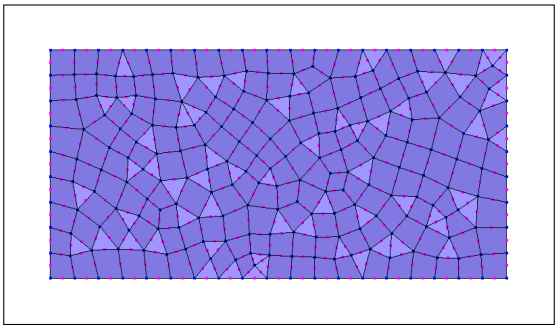


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**, **SELECT**, **'TXT'**, **DEPLA_2D**, **CONTR_2D**, **REAC_2D**
- Etude : **lancer**

5.3.12 Test sslp001l

Allure du maillage.

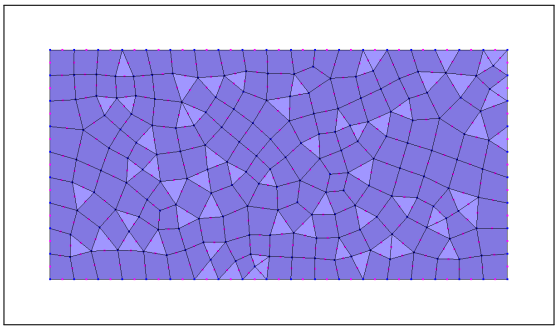


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'NEWTON_COMPLET'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.13 Test sslp001m

Allure du maillage.

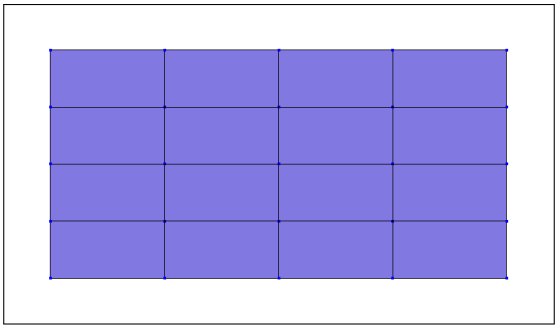


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'NEWTON_COMPLET'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer**

5.3.14 Test sslp002a

Allure du maillage.

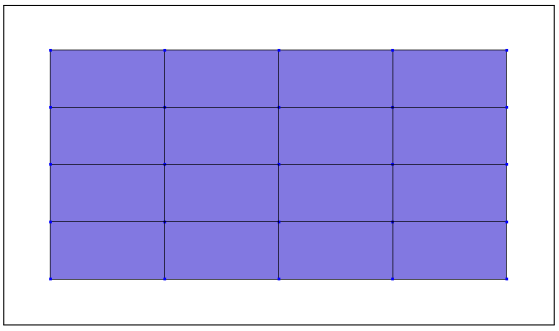


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.15 Test sslp002b

Allure du maillage.

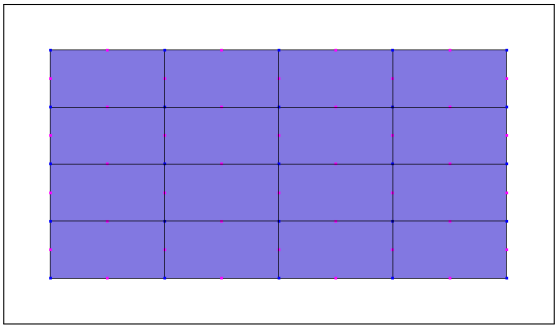


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.16 Test sslp002c

Allure du maillage.

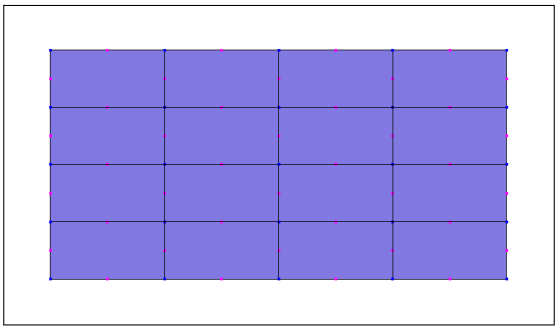


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.17 Test sslp002d

Allure du maillage.

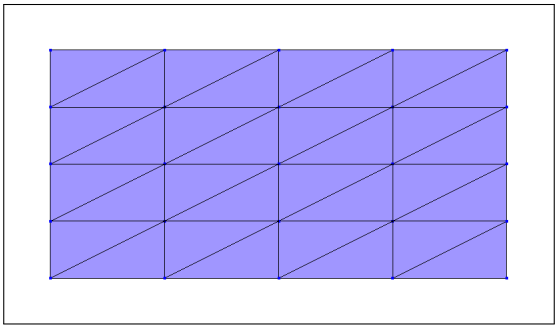


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.18 Test sslp002e

Allure du maillage.

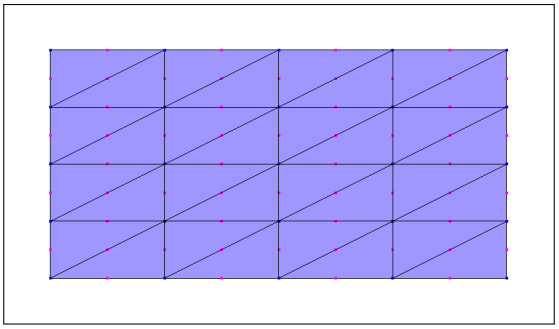


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.19 Test sslp002f

Allure du maillage.

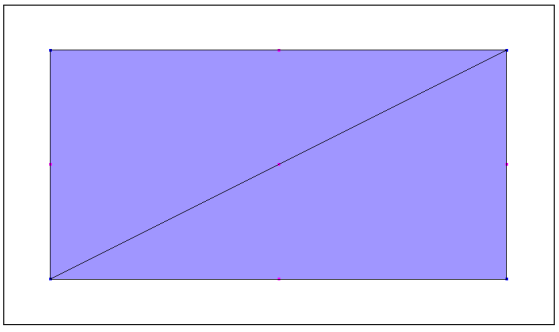


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.20 Test sslp003

Allure du maillage.

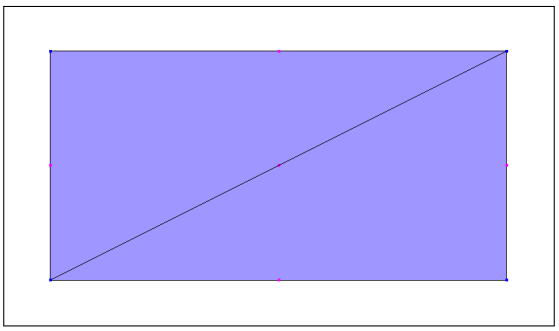


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.3.21 Test sslp004a

Allure du maillage.

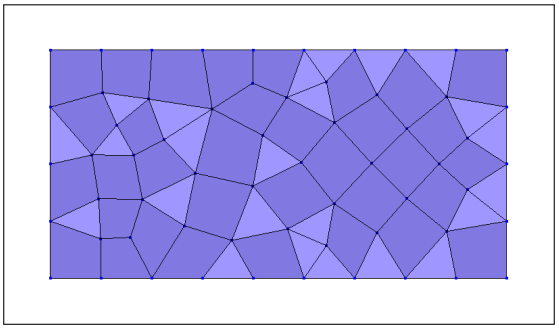


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.3.22 Test sslp005a

Allure du maillage.

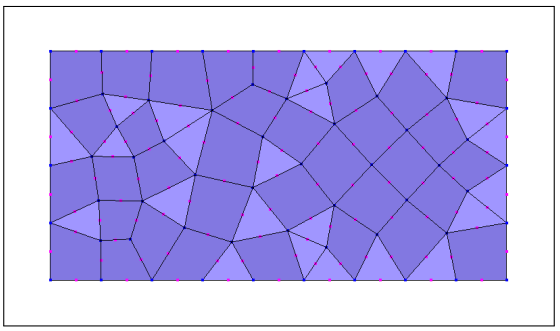


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.23 Test sslp005b

Allure du maillage.

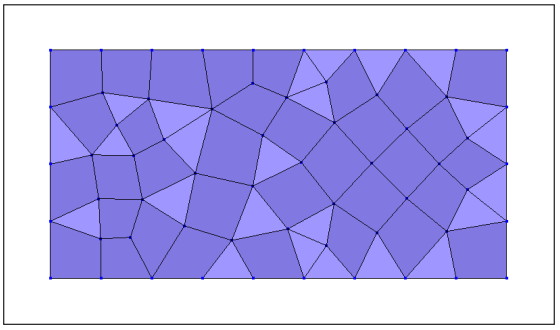


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.24 Test sslp006a

Allure du maillage.

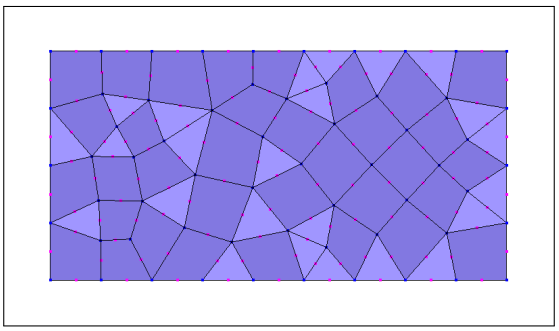


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.3.25 Test sslp006b

Allure du maillage.

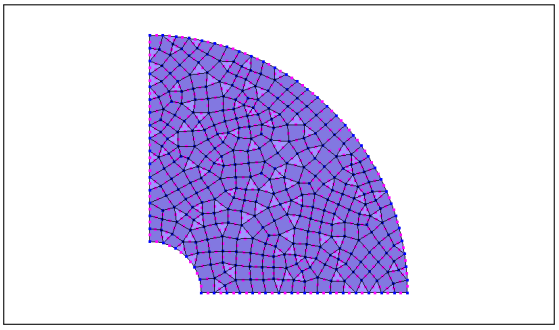


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.3.26 Test sslp007a

Allure du maillage.

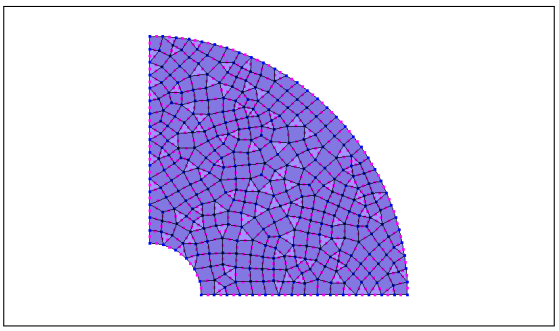


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_ARETE_2D**, **CISAIL_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.27 Test sslp008a

Allure du maillage.

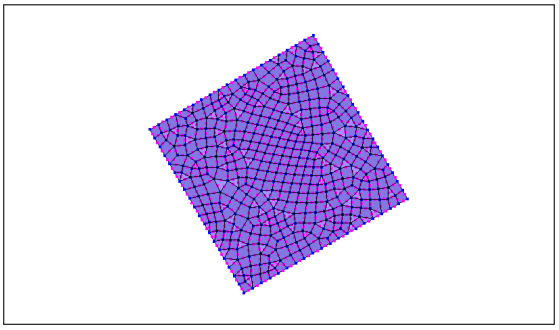


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_ARETE_2D**, **CISAIL_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.3.28 Test sslp009a

Allure du maillage.

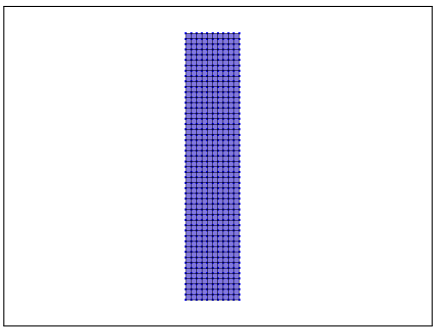


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**, **BASE_2D**
- Chargements : **PRESS_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.29 Test sslp010

Allure du maillage.

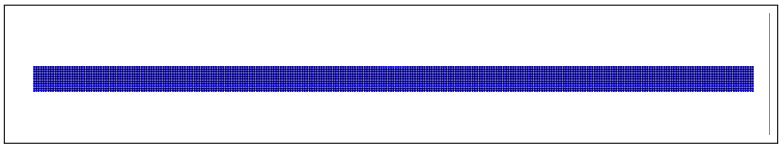


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **GEN_IIP**, **GEN_IRC**
- Eléments : **MB**, **MB_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PNU**, **CHAR_CNU CHAR_PHS**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.3.30 Test sslp011a

Allure du maillage.

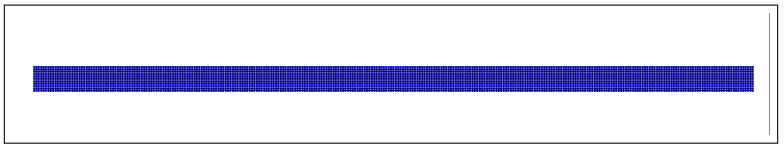


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **BAR_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.3.31 Test sslp011b

Allure du maillage.

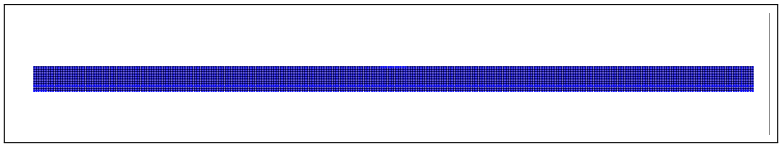


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **BAR_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer**

5.3.32 Test sslp011c

Allure du maillage.



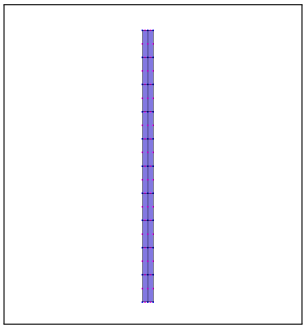
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **BAR_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer_aster**

5.3.33 Test sslp012

Ce test correspond au test de non regression "colo2_axif1" de CESAR.

Allure du maillage.

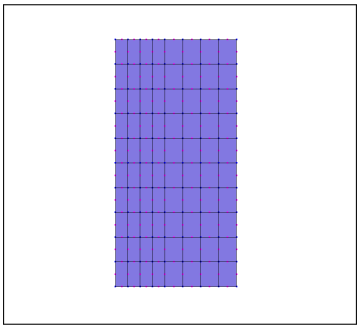


Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **GEN_IIP**, **GEN_IRC**
- Eléments : **AX**, **AX_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PNU**
- Calcul : **AXIF**
- Jeu de données : **lancer**

5.3.34 Test sslp013

Ce test correspond au test de non regression "colba_sp2r" de CESAR.

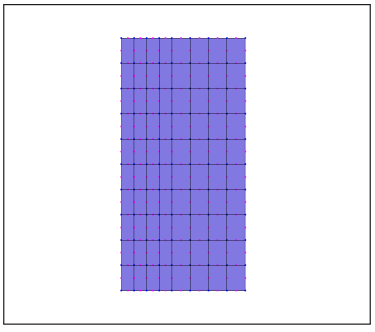
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Utilitaires :
- Eléments : **MB**, **MB_ELI**, **SP**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.3.35 Test sslp014

Allure du maillage.

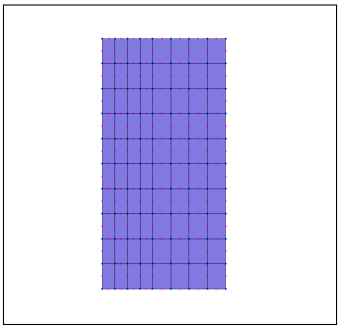


Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**

— Modèles : **FORMUL_MECA**, **'AXISYM'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques :
— Liaisons : **MATR_ELEM**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **PRESS_ARETE_2D**
— Résolutions : **STAT_LINE**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.3.36 Test sslp015

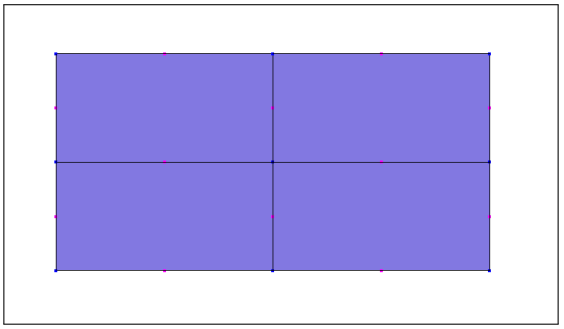
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'AXISYM'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques :
— Liaisons : **RELA_LINE**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **PRESS_ARETE_2D**
— Résolutions : **STAT_LINE**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.3.37 Test sslp016

Allure du maillage.



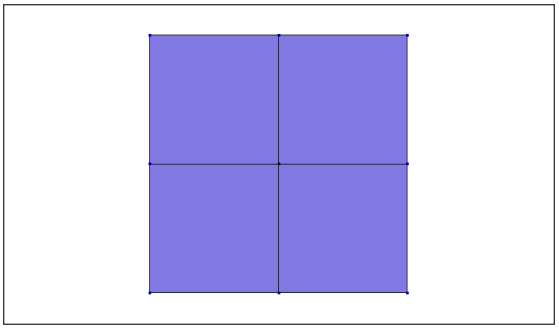
Mots-clés.

- Import Python : modele_donnees (langage Pilote), libMEDMEM_Swig (langage MED Mémoire)
- Maillages : **MESHING**, **setMesh**, **setIntToExtEntites**, **setExtToIntEntites**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4 Catégorie "structure statique linéaire surfacique" (ssls)

5.4.1 Test ssls001

Allure du maillage.

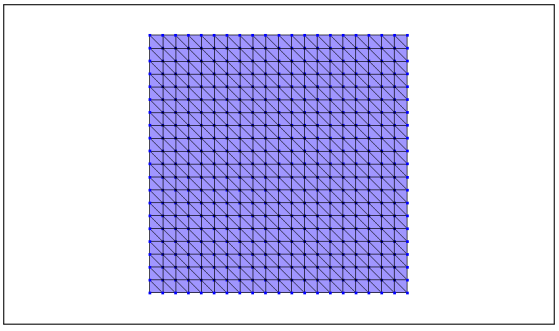


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_STD_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.4.2 Test ssls002a

Allure du maillage.

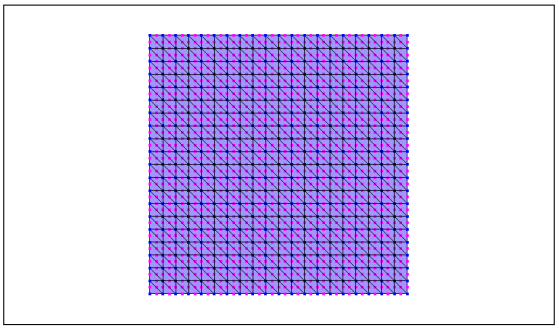


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MINC'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.3 Test ssls002b

Allure du maillage.

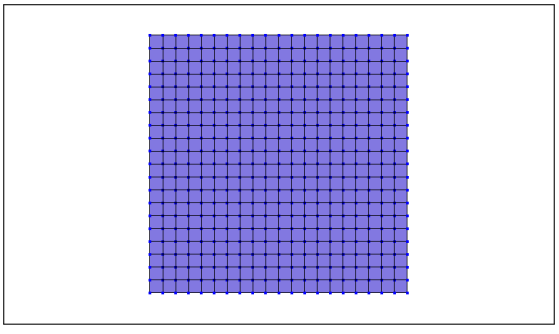


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_EPAIS'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.4 Test ssls002c

Allure du maillage.

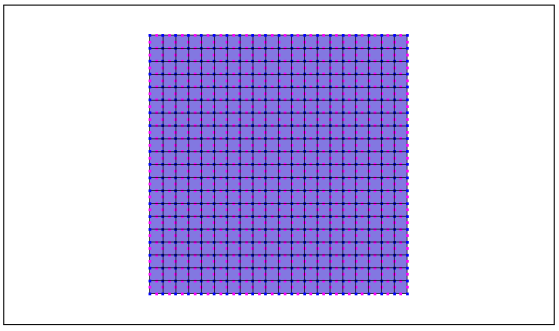


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MINC'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.5 Test ssls002d

Allure du maillage.

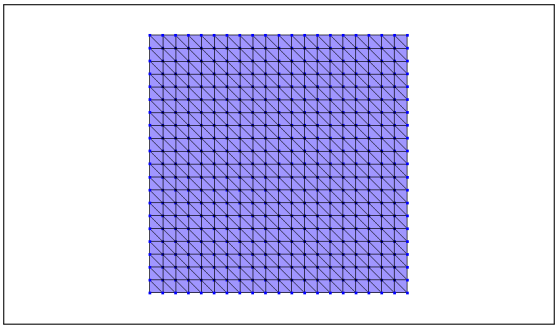


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_EPAIS'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.6 Test ssls003a

Allure du maillage.

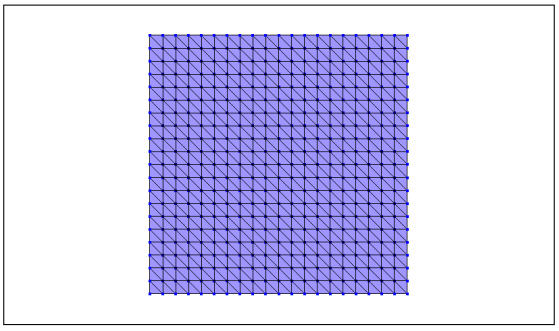


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MINC'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.4.7 Test ssls003b

Allure du maillage.

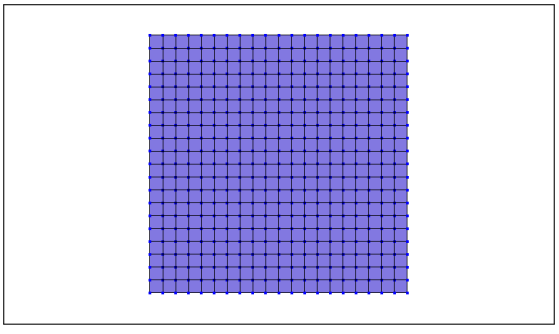


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_EPAIS'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.4.8 Test ssls003c

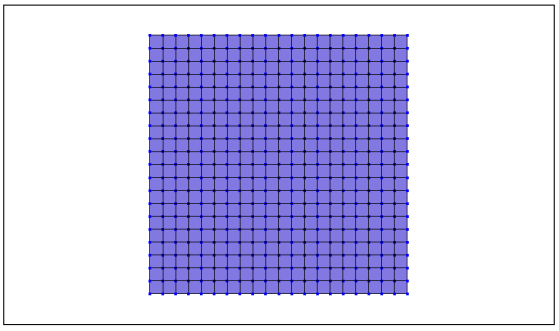
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MINC'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.4.9 Test ssls003d

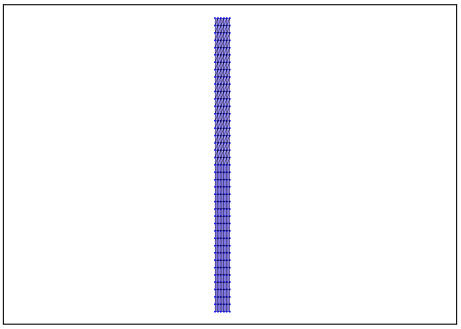
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_EPAIS'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.4.10 Test ssls004a

Allure du maillage.

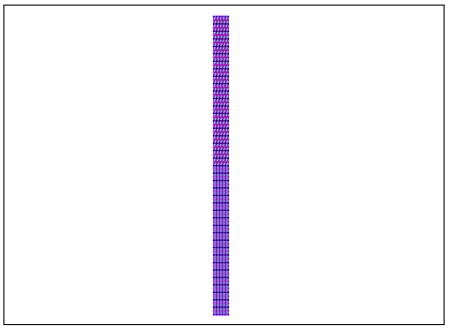


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.11 Test ssls004b

Allure du maillage.

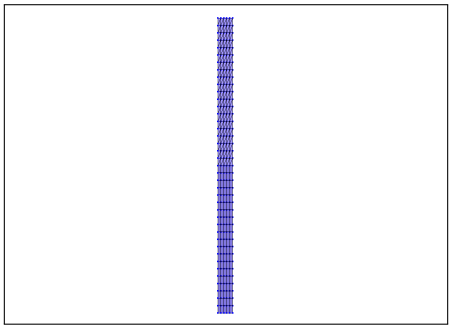


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.12 Test ssls005a

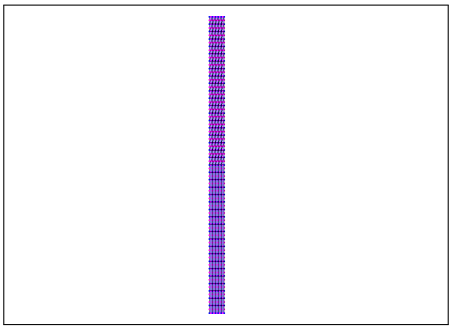
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.4.13 Test ssls005b

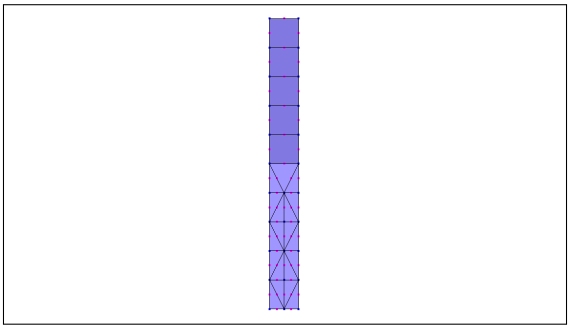
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.4.14 Test ssls006

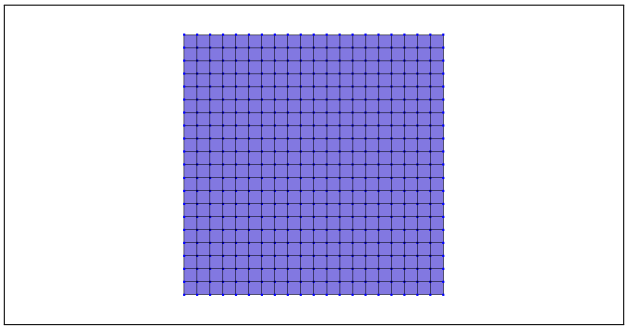
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_MC**, **CO_LC**, **CO_LC_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.4.15 Test ssls007a

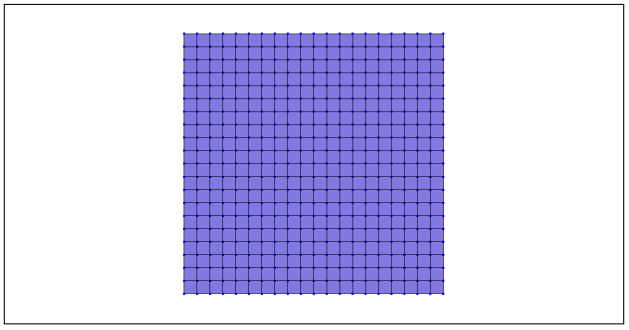
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MULTI'**
- Matériaux : **COMPOSITE**, **'MULTI_COUCH'**, **ELAS_LINE_ISO**
- Caractéristiques : **COQUE_MULTI**, **GROUP_COUCHE**, **COUCHE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.16 Test ssls008a

Allure du maillage.

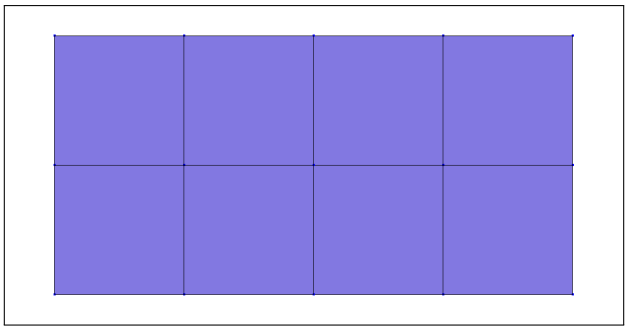


Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**

— Modèles : **FORMUL_MECA**, **'COQ_MULTI'**
— Matériaux : **COMPOSITE**, **'MULTI_COUCH'**, **ASTER_ELAS_LINE_ORTHO**
— Caractéristiques : **COQUE_MULTI**, **GROUP_COUCHE**, **COUCHE**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_MAILLE_COQ**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.4.17 Test ssls009a

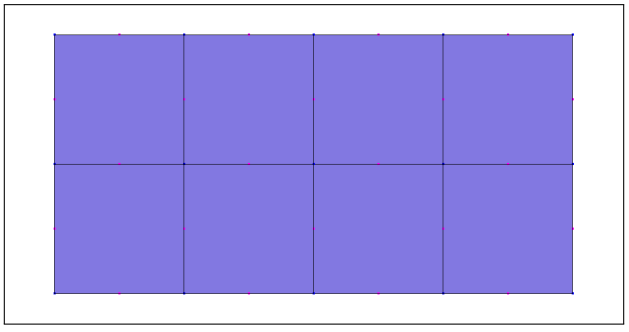
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **CO**, **CO_STD_ELI**, **RL**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_PUR**
— Calcul : **LINE**
— Jeu de données : **lancer**

5.4.18 Test ssls009b

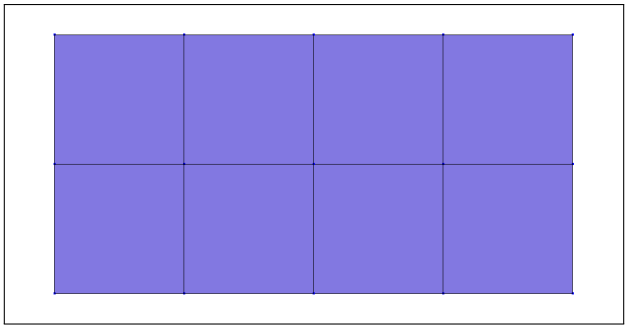
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **CO**, **CO_STD_ELI**, **RL**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_PUR**
— Calcul : **LINE**
— Jeu de données : **lancer**

5.4.19 Test ssls009c

Allure du maillage.

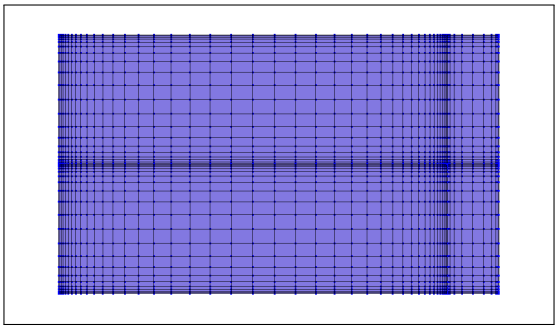


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_STD_ELI**, **SP**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.4.20 Test ssls010a

Allure du maillage.

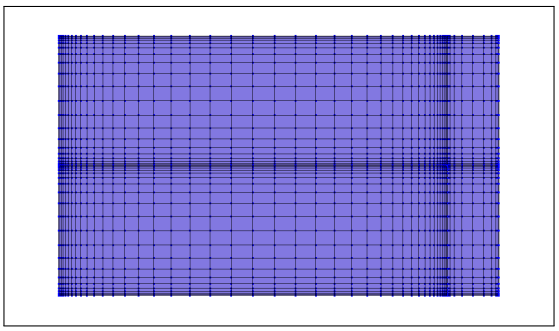


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MINC'**, **'POUT_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**, **POUT_3D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.4.21 Test ssls010b

Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'COQ_MINC'**, **'POUT_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**, **POUT_3D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_MAILLE_COQ**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer**

5.4.22 Test ssls011

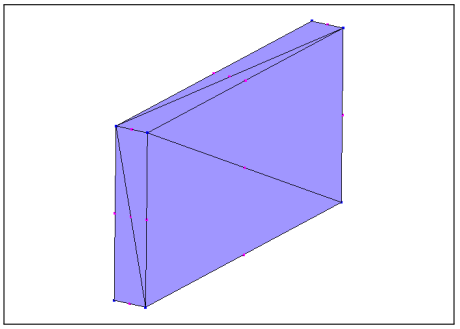
Allure du maillage.

Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **EXPO**, **GEFI**
- Eléments : **CO**, **CO_STD_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **LINE**
- Jeu de données : **lancer**
- Résultats : **RSV4**

5.5 Catégorie "structure statique linéaire volumique" (sslv)

5.5.1 Test sslv001a

Allure du maillage.

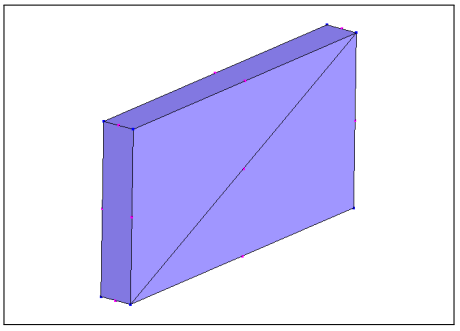


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.2 Test sslv001b

Allure du maillage.

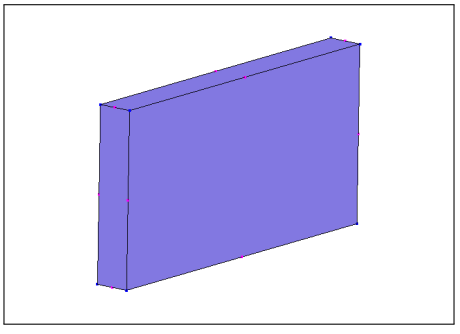


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.3 Test sslv001c

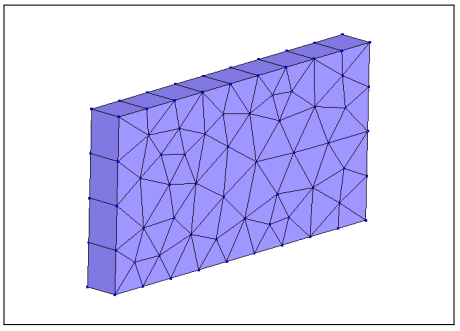
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.4 Test sslv001d

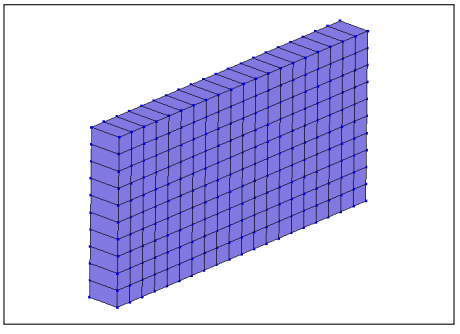
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.5 Test sslv001e

Allure du maillage.

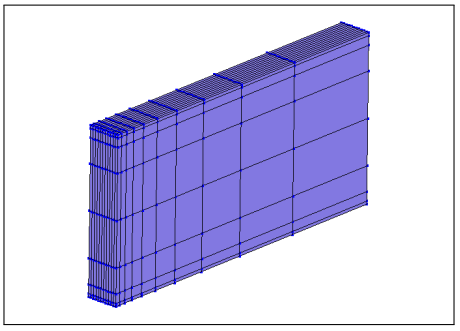


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.6 Test sslv001f

Allure du maillage.

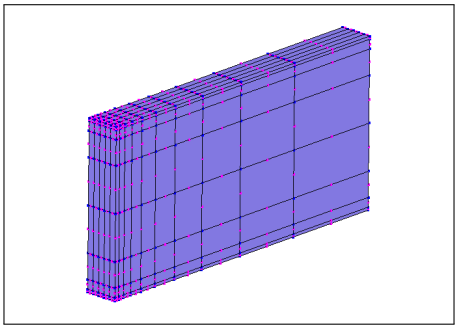


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.7 Test sslv001g

Allure du maillage.

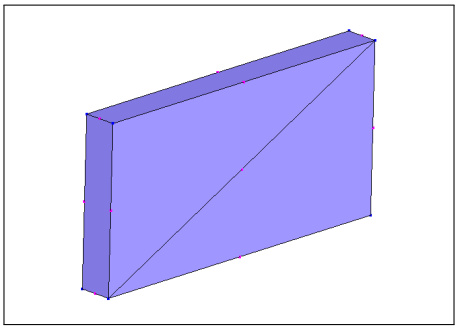


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.8 Test sslv001h

Allure du maillage.

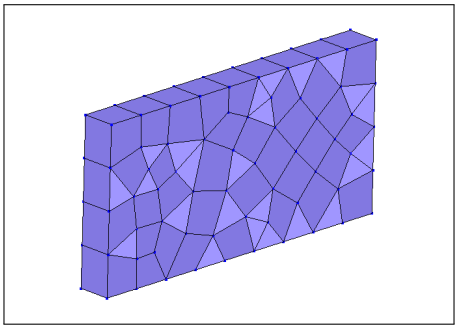


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.9 Test sslv002a

Allure du maillage.

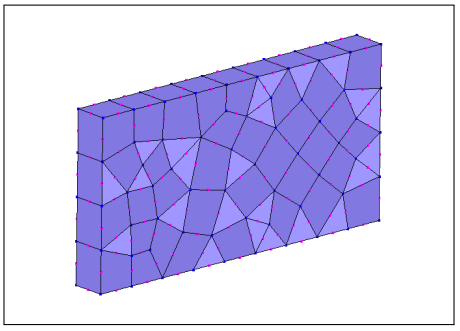


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.5.10 Test sslv002b

Allure du maillage.

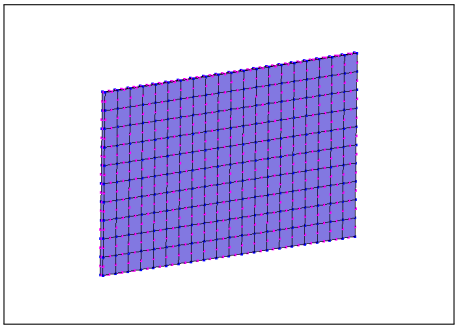


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.5.11 Test sslv003a

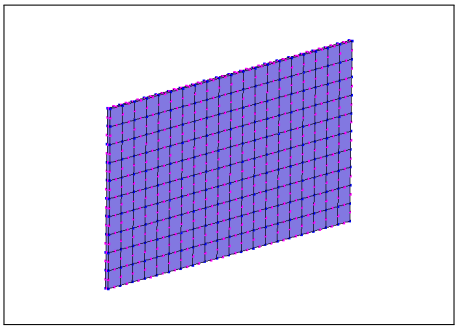
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **MODAL_LINE**, **'FLAMB'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.12 Test sslv004a

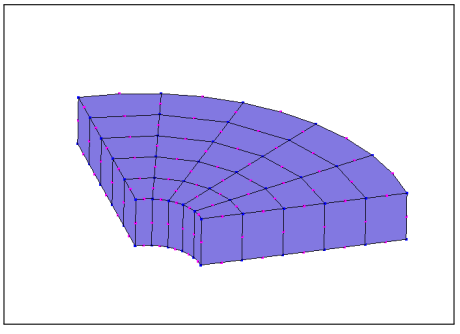
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_FACE_3D**
- Résolutions : **MODAL_LINE**, **'FLAMB'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.5.13 Test sslv005a

Allure du maillage.

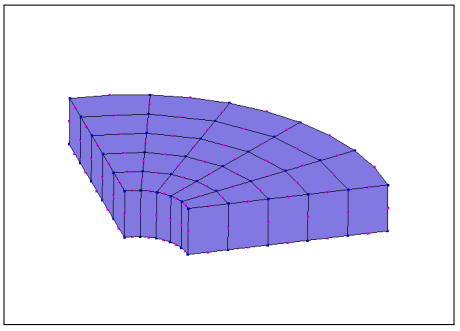


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.14 Test sslv006a

Allure du maillage.

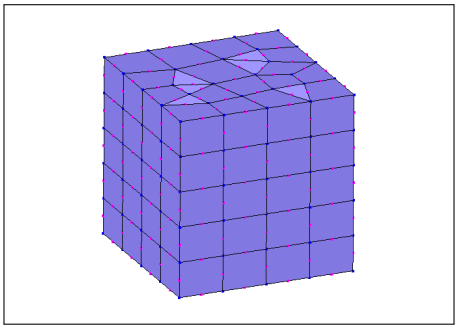


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.5.15 Test sslv007a

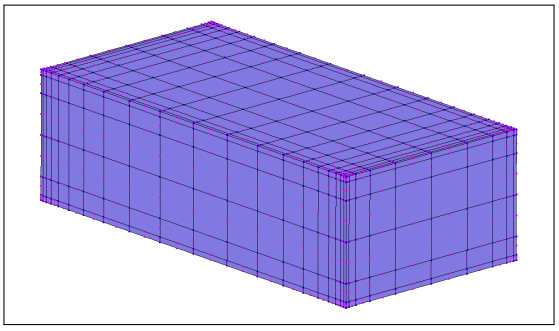
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**, **BASE_3D**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.16 Test sslv008a

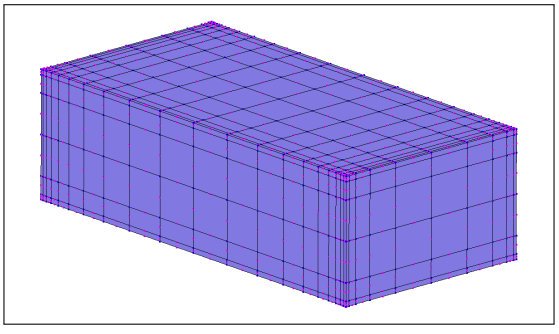
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**, **'COQ_EPAIS'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.5.17 Test sslv008b

Allure du maillage.

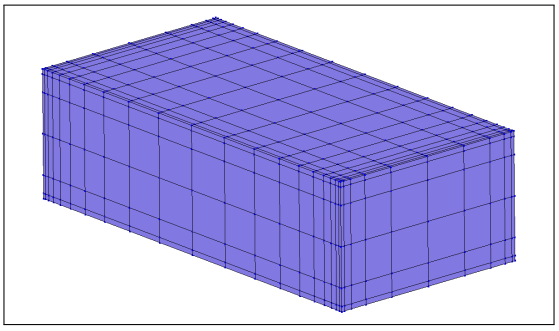


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**, **'COQ_EPAIS'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer**

5.5.18 Test sslv008c

Allure du maillage.

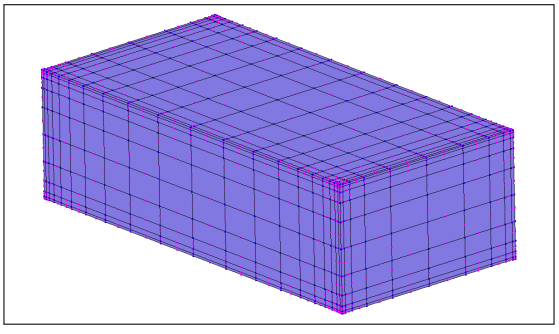


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**, **'COQ_EPAIS'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer_aster**

5.5.19 Test sslv008d

Allure du maillage.

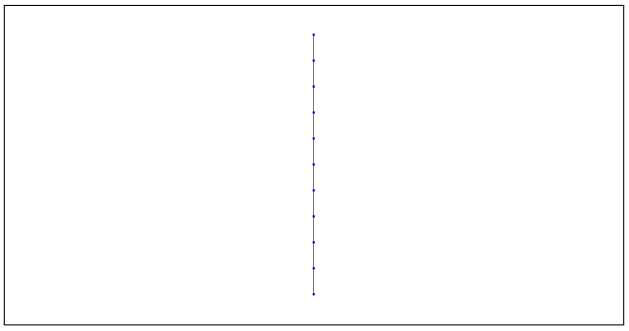


Mots-clés.
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**, **'COQ_EPAIS'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **COQUE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
- Résultats : **FICHIER**, **'MED'**
- Etude : **lancer**

5.6 Catégorie "structure statique non linéaire linéique" (ssnl)

5.6.1 Test ssnl001a

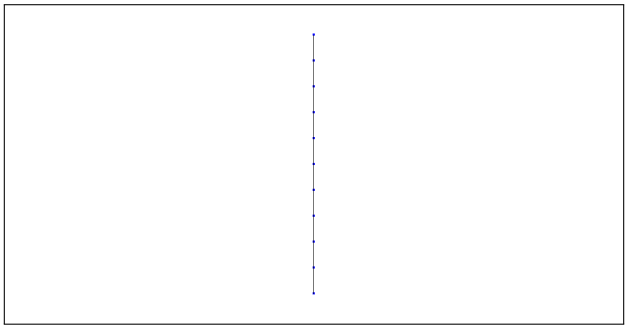
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **PT**, **PT_MF**, **PT_LF**, **PT_LF_VMSE**, **PT_CF**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.6.2 Test ssnl001b

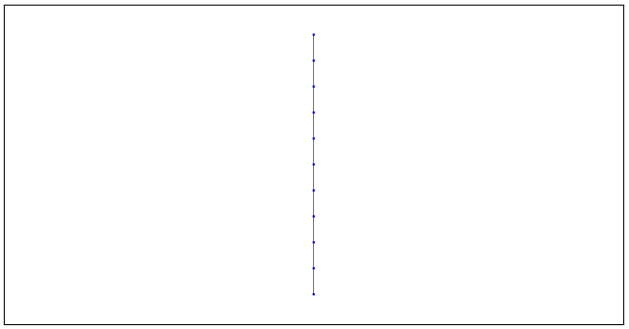
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **PT**, **PT_MF**, **PT_LF**, **PT_LF_VMAE**, **PT_CF**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.6.3 Test ssnl001c

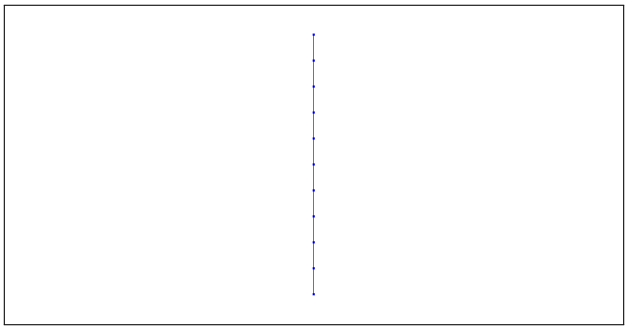
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **PT**, **PT_MF**, **PT_LF**, **PT_LF_CP**, **PT_CF**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.6.4 Test ssnl001d

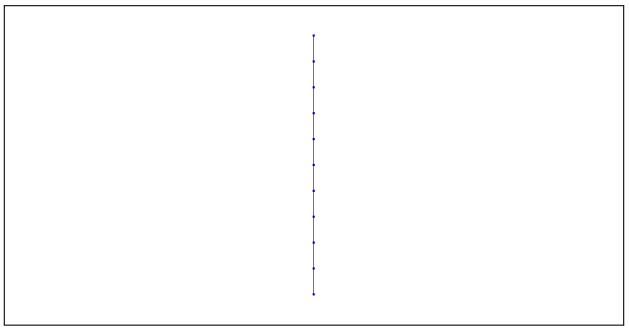
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **PT**, **PT_MF**, **PT_LF**, **PT_LF_WWS**, **PT_CF**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.6.5 Test ssnl001e

Allure du maillage.



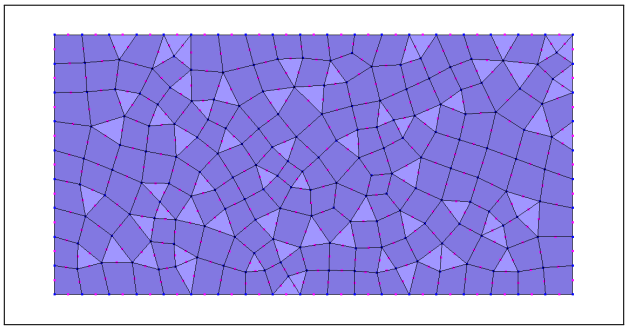
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **PT**, **PT_MF**, **PT_LF**, **PT_LF_WWM**, **PT_CF**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7 Catégorie "structure statique non linéaire plane" (ssnp)

5.7.1 Test ssnp001a

Allure du maillage.

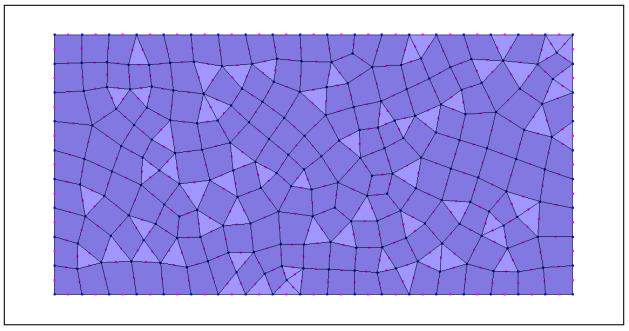


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **VON_MISES_AVEC_ECROU**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'NEWTON_COMPLET'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.2 Test ssnp002a

Allure du maillage.

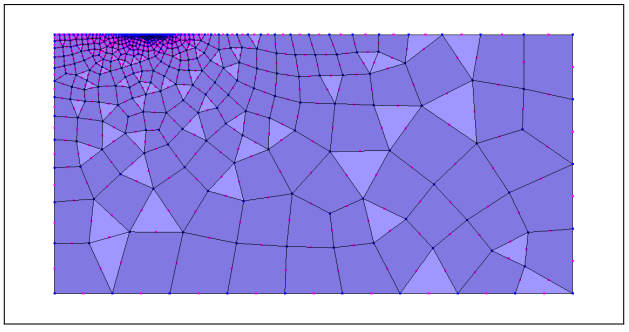


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
- Matériaux : **ASTER_VMIS_ISOT_LINE**, **ELAS**, **ECRO_LINE**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'NEWTON_COMPLET'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.7.3 Test ssnp003a

Allure du maillage.



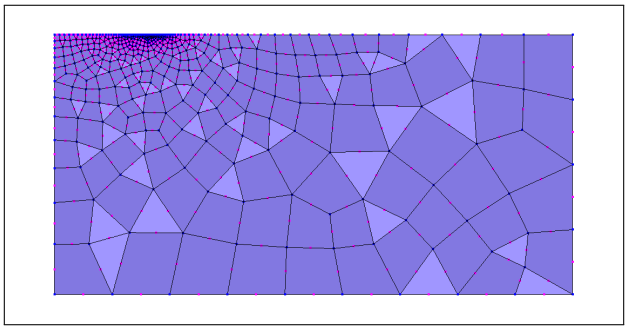
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**, **MOHR_COUL_SANS_ECROU**, **VON_MISES_SANS_ECROU**, **VON_MISES_AVEC**
  DRUCK_PRAG_SANS_ECROU, **DRUCK_PRAG_AVEC_ECROU**, **CRIT_PARABOL**, **CRIT_ORIENT**, **HOEK_BROWN**, **TRESCA_ANISO**, **ELAS_DILAT_ISO**, **CAM_CLAY_MODIF**, **PREVOST_HOEG**, **ELAS_LINE_ORTHO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **POIDS**, **FORC_ARETE_2D**

— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.7.4 Test ssnp003b

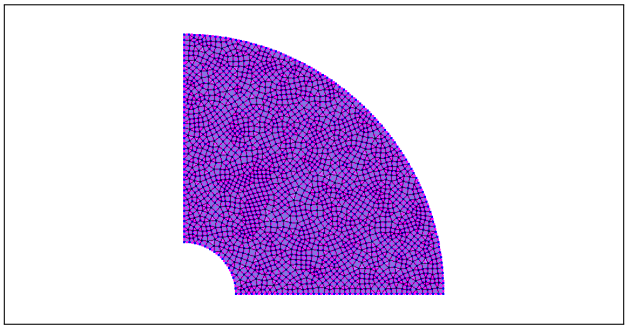
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
— Matériaux : **MATER_RENFORCE**, **'BUHAN_SUDRET'**, **DRUCK_PRAG_SANS_ECROU**, **ELASTO_PLAST_1D**
— Caractéristiques : **DEFOR_PLANE**, **RENFOR_DEFOR_PLANE**, **'HOMOGEN'**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **POIDS**, **FORC_ARETE_2D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.7.5 Test ssnp004a

Allure du maillage.

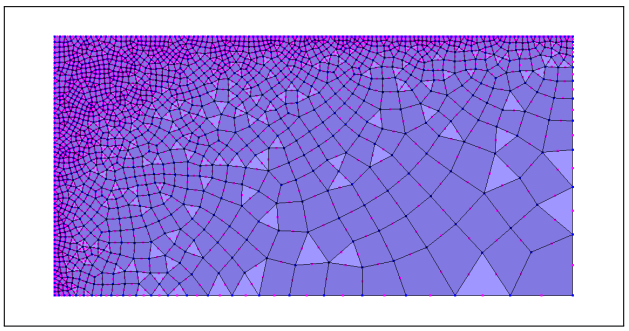


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**

— Matériaux : **TRESCA_ANISO**, **MATER_RENFORCE**, **'BUHAN_SUDRET'**, **DRUCK_PRAG_SANS_ECROU**, **ELASTO_PLAST_1D**
— Caractéristiques : **DEFOR_PLANE**, **RENFOR_DEFOR_PLANE**, **'RADIAL'**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **PRESS_ARETE_2D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.7.6 Test ssnp005a

Allure du maillage.

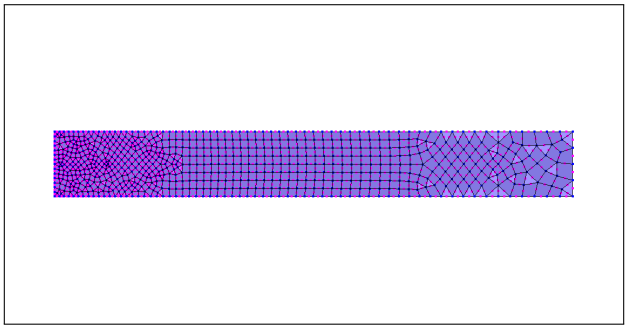


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_CPLAN'**
— Matériaux : **ELAS_LINE_ISO**, **VON_MISES_SANS_ECROU**, **VON_MISES_AVEC_ECROU**, **CRIT_PARABOL**, **CRIT_ORIENT**, **ELAS_DILAT_ISO**
— Caractéristiques : **CONTR_PLANE**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **POIDS**, **FORC_ARETE_2D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.7.7 Test ssnp006a

Allure du maillage.

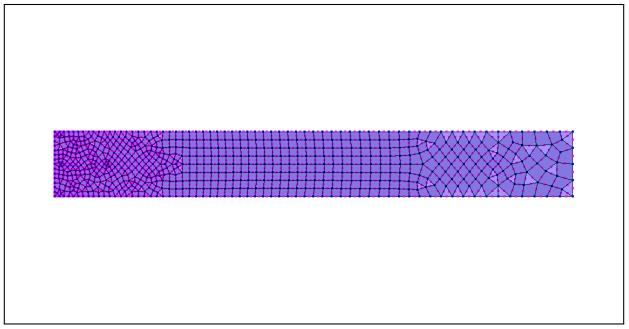


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **ELAS_LINE_ISO**, **MOHR_COUL_SANS_ECROU**, **VON_MISES_SANS_ECROU**, **VON_MISES_AVEC**
  DRUCK_PRAG_SANS_ECROU, **DRUCK_PRAG_AVEC_ECROU**, **CRIT_PARABOL**, **CRIT_ORIENT**,
  HOEK_BROWN, **ELAS_DILAT_ISO**, **CAM_CLAY_MODIF**, **PREVOST_HOEG**, **ELAS_LINE_ORTHO**
- Caractéristiques : **CONTR_PLANE**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**,
  INCREMENTATION, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.8 Test ssnp006b

Allure du maillage.

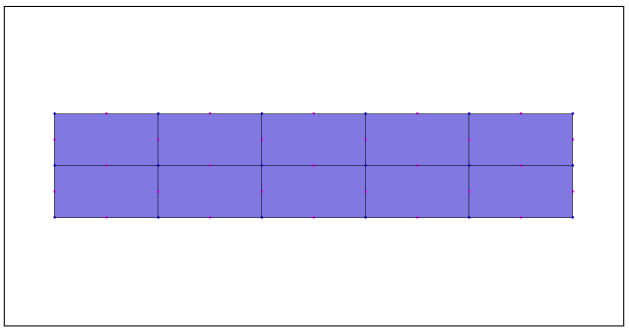


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **MATER_RENFORCE**, **'BUHAN_SUDRET'**, **DRUCK_PRAG_SANS_ECROU**,
  ELASTO_PLAST_1D
- Caractéristiques : **AXISYM RENFOR_AXISYM**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**,
  INCREMENTATION, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.9 Test ssnp007

Allure du maillage.

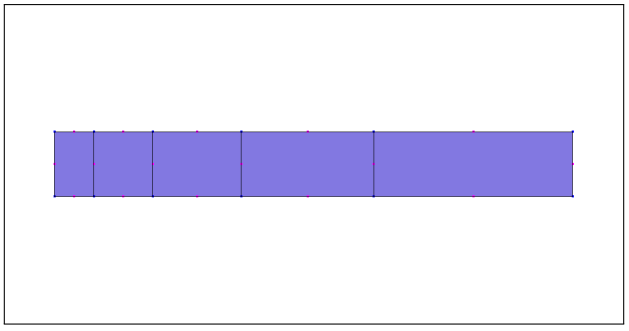


Mots-clés.

— Import Python : modele_cesar (langage CESAR)
— Eléments : **MB**, **MB_ELI**, **FD**, **FD_FC**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_POI**, **CHAR_PUR**
— Calcul : **TCNL**
— Jeu de données : **lancer**

5.7.10 Test ssnp008a

Allure du maillage.

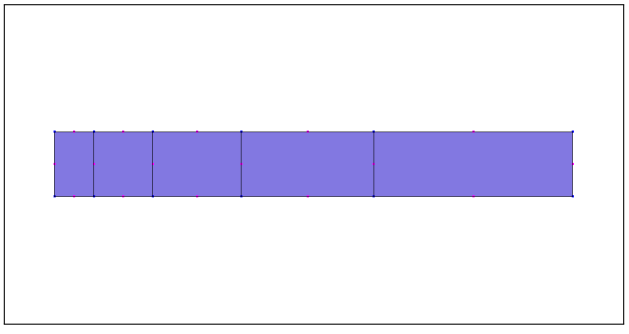


Mots-clés.

— Import Python : modele_cesar (langage CESAR)
— Eléments : **MB**, **MB_VE**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_PUR**
— Calcul : **MCNL**
— Jeu de données : **lancer**

5.7.11 Test ssnp008b

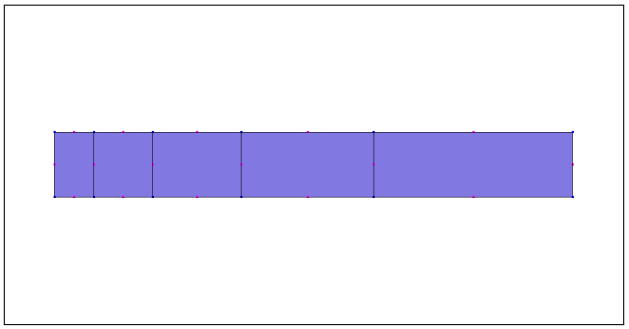
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_NO**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.12 Test ssnp008c

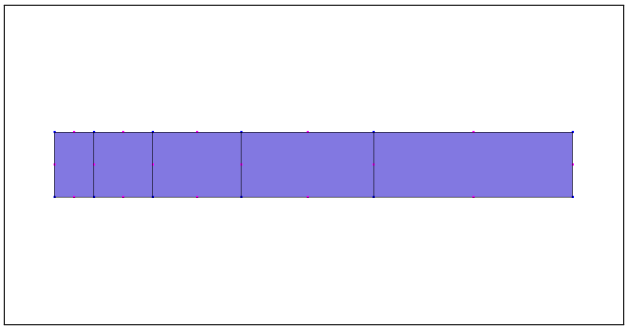
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_WWS**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.13 Test ssnp008d

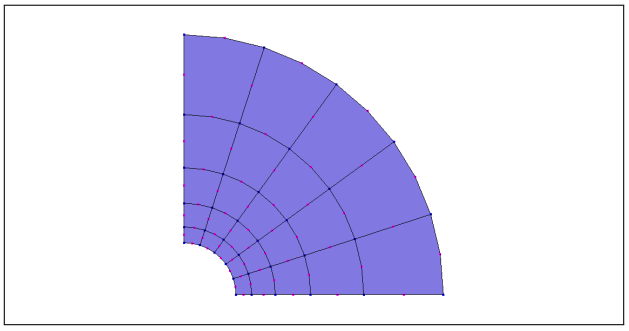
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_WWM**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.14 Test ssnp009a

Allure du maillage.

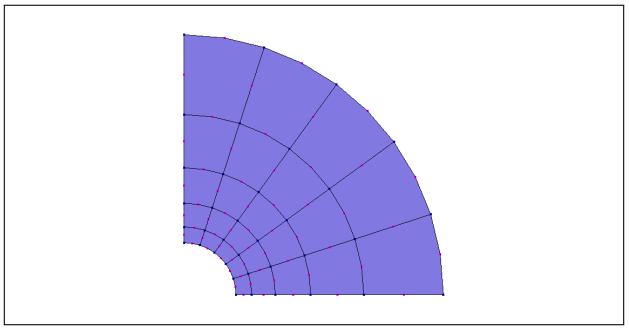


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ME**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.15 Test ssnp009b

Allure du maillage.

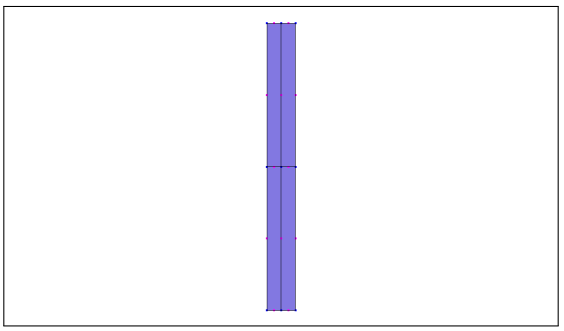


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_MCSE_EO**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.16 Test ssnp010a

Allure du maillage.

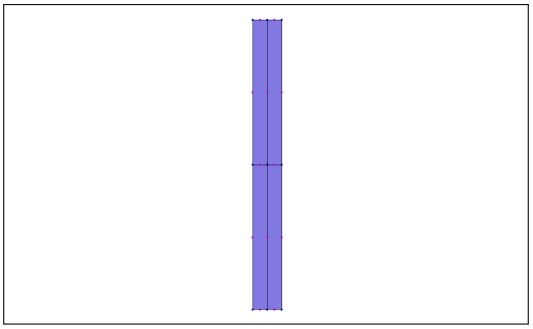


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.17 Test ssnp010b

Allure du maillage.

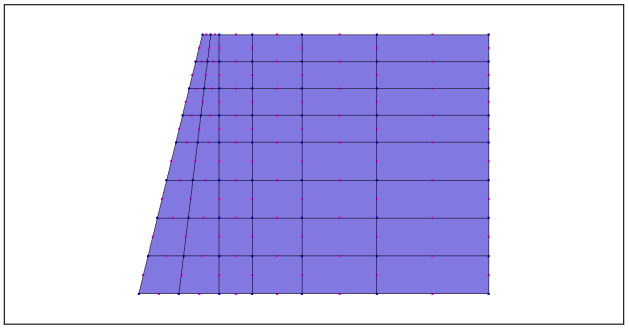


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_EFD**, **EFD_COUC**, **EFD_INST**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.18 Test ssnp011a

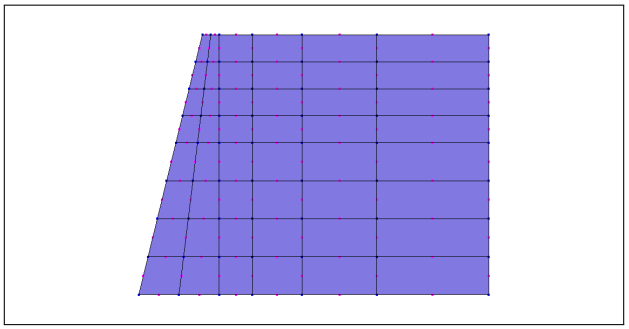
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ELI**, **MB_MCSE**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.7.19 Test ssnp011b

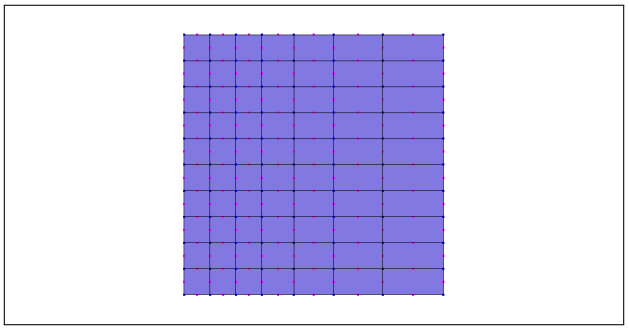
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ELI**, **MB_MCSE**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_LAM**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.7.20 Test ssnp012

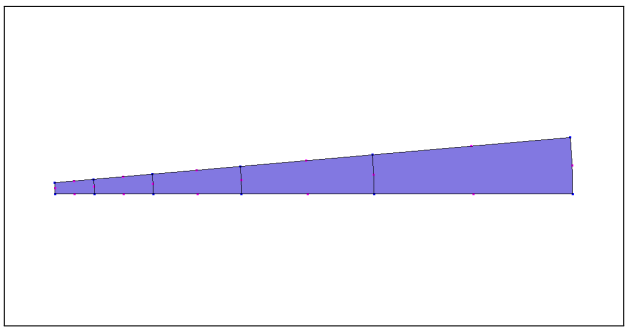
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_ELI**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SIG**, **SIG_GEO**, **SIG_COUC**, **CHAR_POI**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.7.21 Test ssnp013

Allure du maillage.

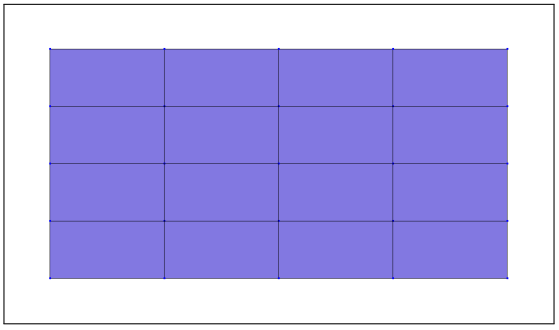


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_MCSE**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_SIG**, **SIG_CST**, **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.22 Test ssnp014a

Allure du maillage.

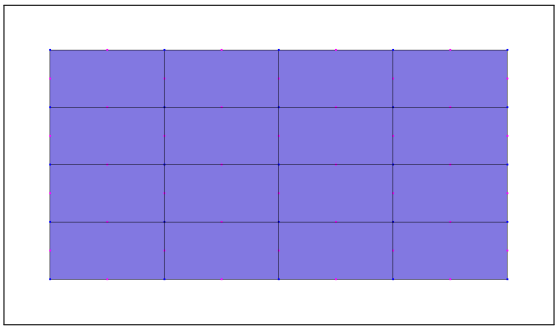


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.23 Test ssnp014b

Allure du maillage.

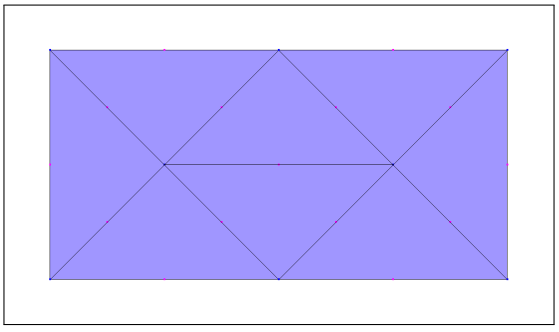


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.24 Test ssnp014c

Allure du maillage.



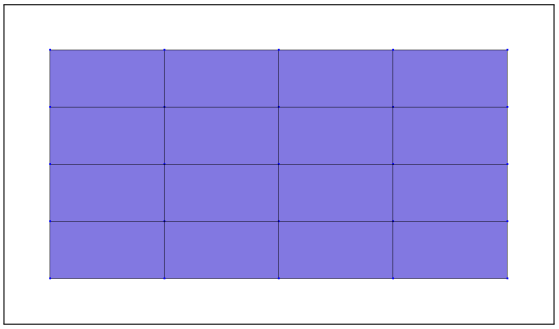
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**

— Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.7.25 Test ssnp014d

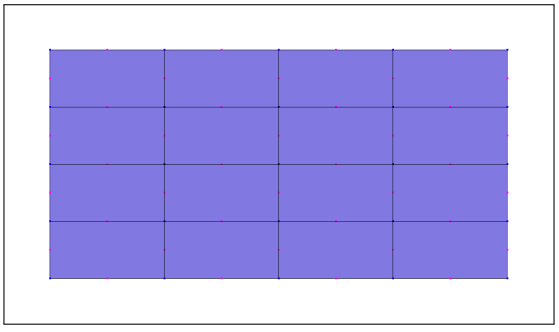
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
— Matériaux : **ASTER_ELAS**
— Caractéristiques :
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_ARETE_2D**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.7.26 Test ssnp014e

Allure du maillage.

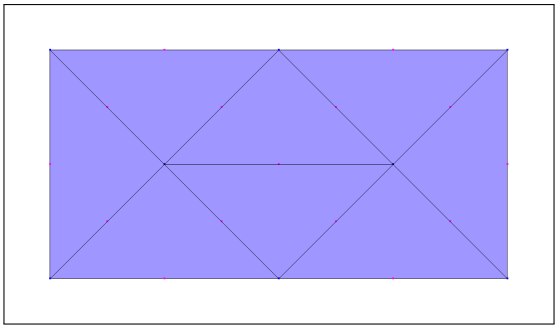


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
— Matériaux : **ASTER_ELAS**
— Caractéristiques :

— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_ARETE_2D**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.7.27 Test ssnp014f

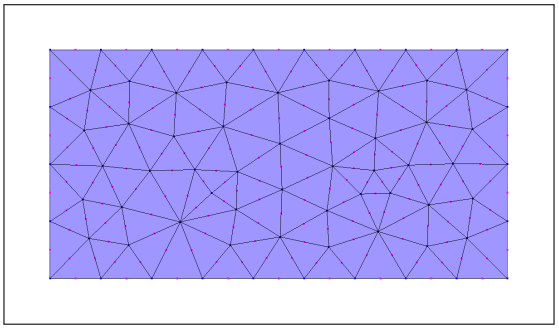
Allure du maillage.

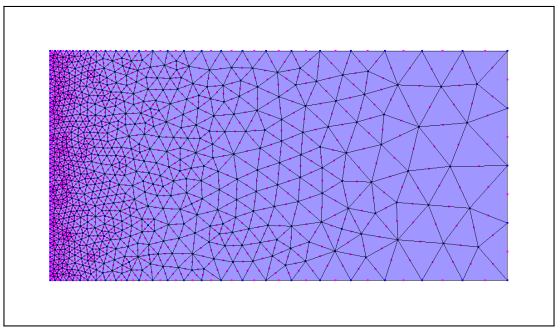


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
— Matériaux : **ASTER_ELAS**
— Caractéristiques :
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_ARETE_2D**
— Résolutions : **STAT_LINE**, **'MULTI_FRONT'**
— Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer_aster**

5.7.28 Test ssnp014g

Allure du maillage.



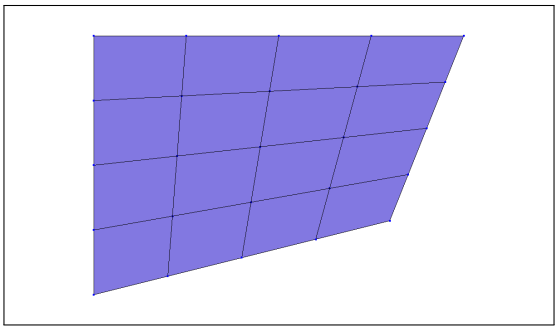


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.29 Test ssnp015a

Allure du maillage.

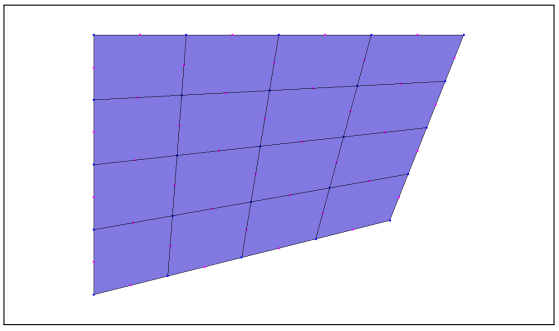


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.30 Test ssnp015b

Allure du maillage.



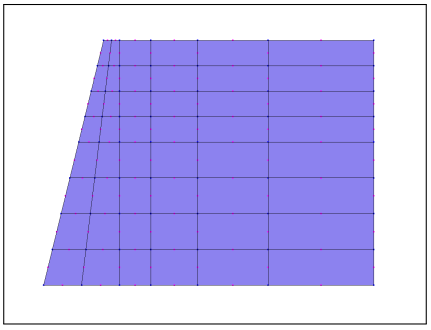
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Post-traitements : **ESTIM_ERREUR**, **'ZZ2'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.31 Test ssnp016a

Ce test correspond au test de non regression "mur_tcnp1" de CESAR.

Allure du maillage.



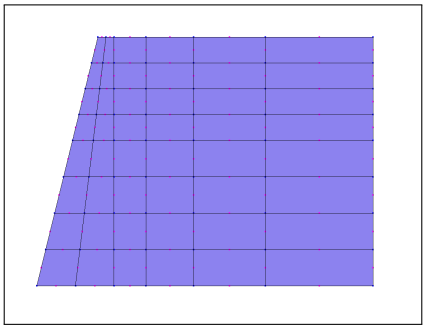
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **ILIG**, **COMT**, **EXPO**, **IMPR**
- Eléments : **MB**, **MB_ELI**, **MB_MCSE**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.7.32 Test ssnp016b

Ce test correspond au test de non regression "mur_tcnp2" de CESAR.

Allure du maillage.



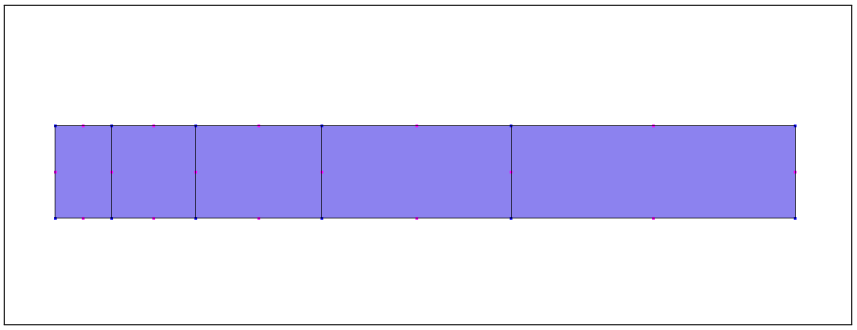
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **ILIG**, **COMT**, **EXPO**, **IMPR**
- Eléments : **MB**, **MB_ELI**, **MB_MCSE**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_LAM**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.7.33 Test ssnp017

Ce test correspond au test de non regression "tubax_c43dc" de CESAR.

Allure du maillage.



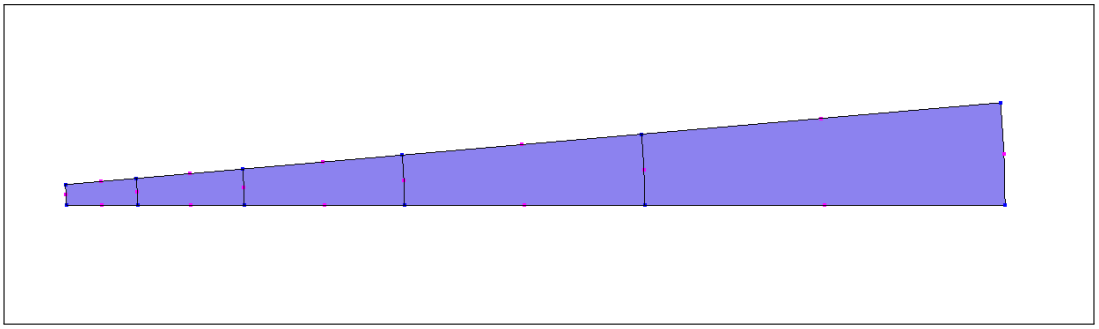
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MB**, **MB_RBS**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.34 Test ssnp018

Ce test correspond au test de non regression "tus2m_ptx4" de CESAR.

Allure du maillage.

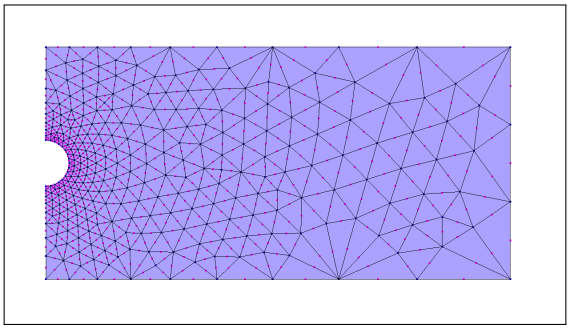


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **EXPO**, **IMPR**
- Eléments : **MB**, **MB_BJA**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements :
- Calcul : **MEXO**, **PTX**
- Jeu de données : **lancer**

5.7.35 Test ssnp020

Allure du maillage.

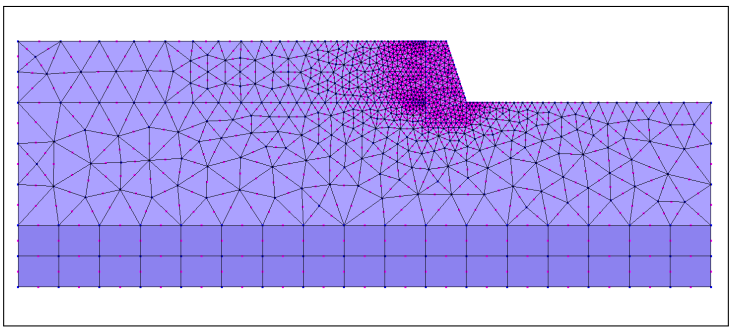


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **EXPO**, **IMPR**
- Eléments : **MB**, **MB_MCSE**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_LAM**, **LAM_COUC**
- Calcul : **MCNL**, **MUL**
- Jeu de données : **lancer**

5.7.36 Test ssnp021

Allure du maillage.

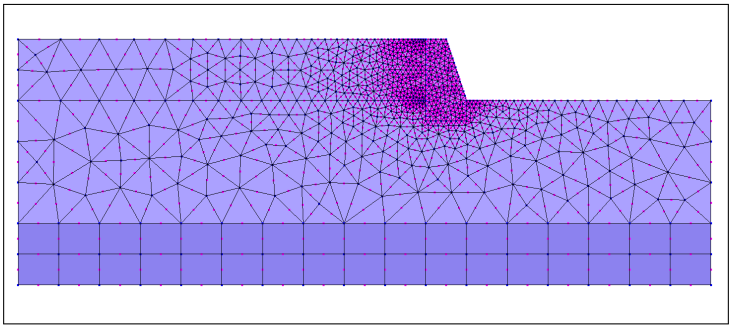


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**, **GFD**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MB**, **MB_MCSE**, **MB_ELI**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SIG**, **SIG_GEO**, **SIG_COUC**, **CHAR_LAM**, **LAM_COUC**, **CHAR_POI**, **POI_FV**
- Calcul : **TCNL**, **MUL**, **STK**, **DPL**, **INI**, **NDP**
- Jeu de données : **lancer**

5.7.37 Test ssnp021b

Allure du maillage.

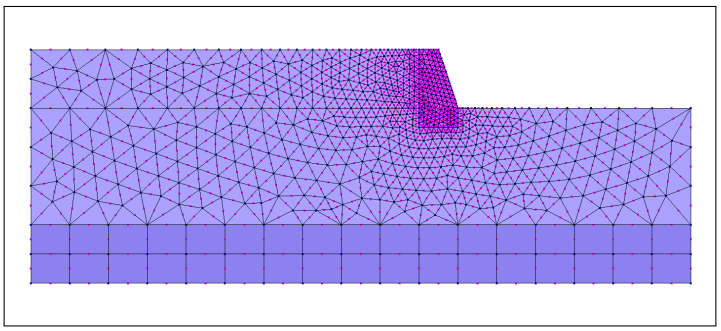


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**, **GFD_GF**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MB**, **MB_MCSE**, **MB_ELI**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SIG**, **SIG_GEO**, **SIG_COUC**, **CHAR_LAM**, **LAM_COUC**, **CHAR_POI**, **POI_FV**
- Calcul : **TCNL**, **MUL**, **STK**, **DPL**, **INI**, **NDP**
- Jeu de données : **lancer**

5.7.38 Test ssnp022

Allure du maillage.



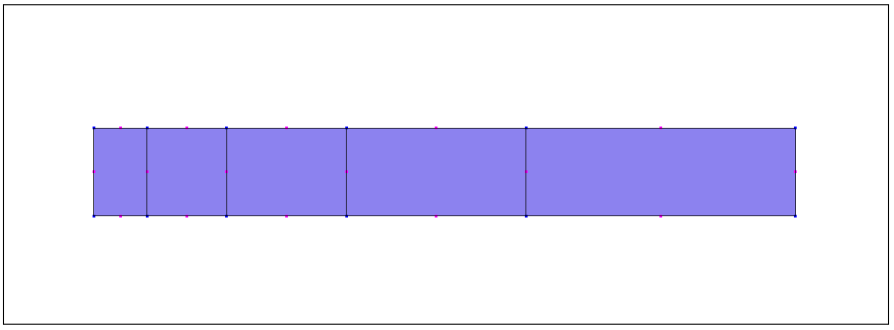
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'CESAR'**, **GFD**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MB**, **MB_MCSE**, **MB_ELI**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SIG**, **SIG_GEO**, **SIG_COUC**, **CHAR_LAM**, **LAM_COUC**, **CHAR_POI**, **POI_FV**
- Calcul : **TCNL**, **MUL**, **STK**, **DPL**, **INI**, **NDP**
- Jeu de données : **lancer**

5.7.39 Test ssnp023

Ce test correspond au test de non regression "tubax_crt16" de CESAR.

Allure du maillage.

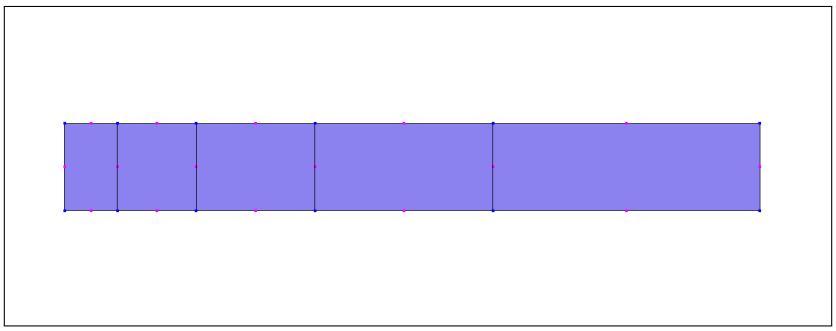


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**
- Eléments : **MB**, **MB_VE**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_PUR**, **CHAR_SIG**, **SIG_CST**,
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.40 Test ssnp024a

Allure du maillage.

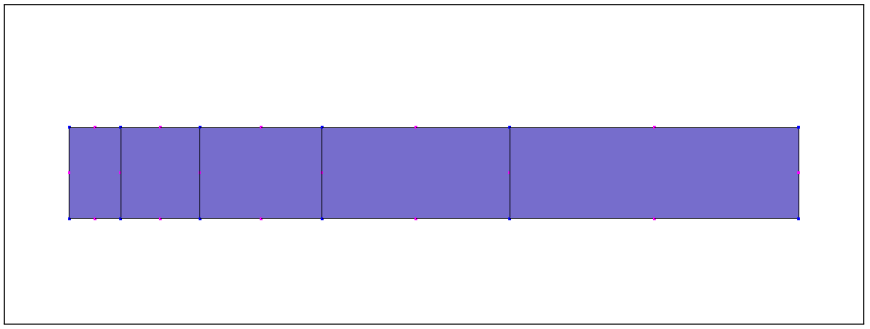


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **VERMEER**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_ARETE_2D**, **CONTR_UNIF**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.41 Test ssnp024b

Allure du maillage.

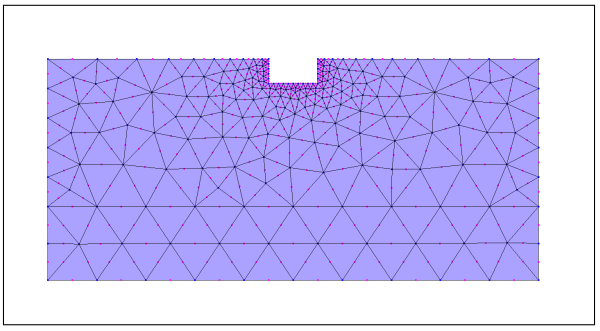


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'AXISYM'**
- Matériaux : **VERMEER**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_ARETE_2D**, **CONTR_UNIF**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.42 Test ssnp025

Allure du maillage.

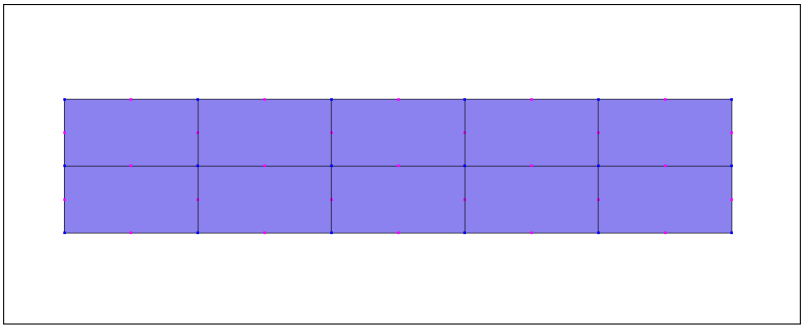


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **MOHR_COUL_SANS_ECROU**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **CONTR_GEOSTAT**, **COUCHE_SOL**, **EXCAV_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.7.43 Test ssnp026

Allure du maillage.



Mots-clés.

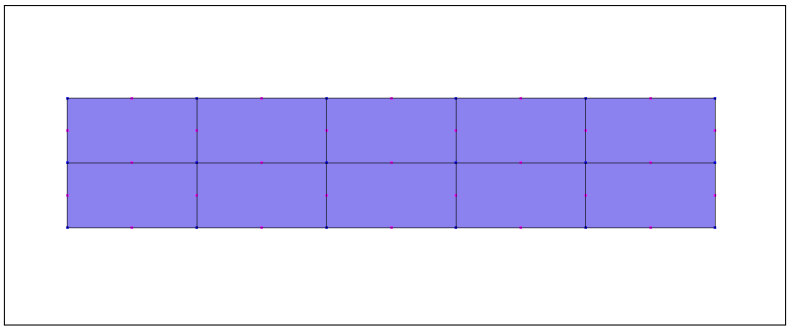
- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **CRIT_PARABOL**
- Caractéristiques :
- Liaisons : **CONTACT**, **FROT_COUL**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **POIDS**, **PRESS_ARETE_2D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**, **VERIF_CONTACT**

— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.7.44 Test ssnp027a

Ce test correspond au test de non regression "bila2_tcnl1" de CESAR.

Allure du maillage.

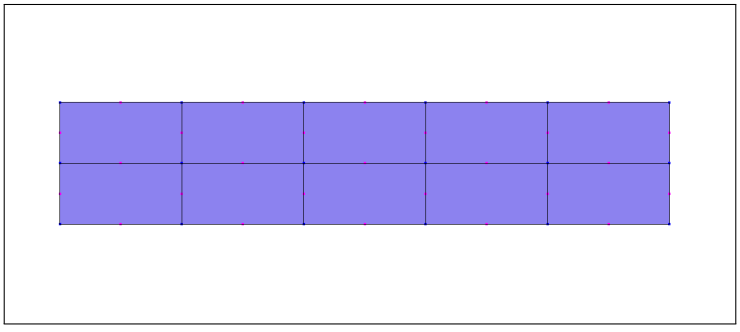


Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Utilitaires : **COMT**, **EXPO**, **IMPR**, **GEN_IIP**, **GEN_IRC**
— Eléments : **MB**, **MB_CP**, **FD**, **FD_FC**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_POI**, **CHAR_PUR**
— Calcul : **TCNL**
— Jeu de données : **lancer**

5.7.45 Test ssnp027b

Ce test correspond au test de non regression "bila2_tcnl1" de CESAR.

Allure du maillage.



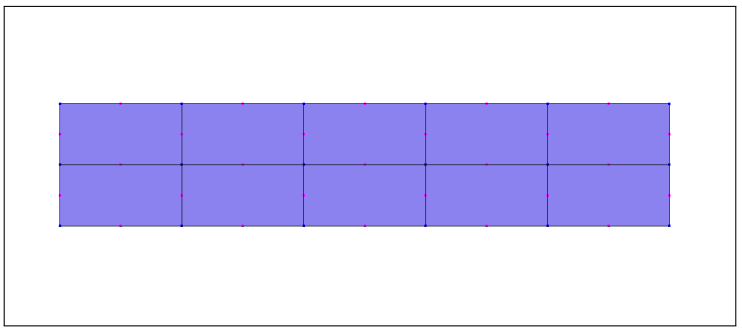
Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Maillage : **MAIL**, **GFD**
— Utilitaires : **COMT**, **EXPO**, **IMPR**, **GEN_IIP**, **GEN_IRC**
— Eléments : **MB**, **MB_CP**, **FD**, **FD_FC**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_POI**, **CHAR_PUR**

— Calcul : **TCNL**
— Jeu de données : **lancer**

5.7.46 Test ssnp027c

Ce test correspond au test de non regression "bila2_tcnl1" de CESAR.

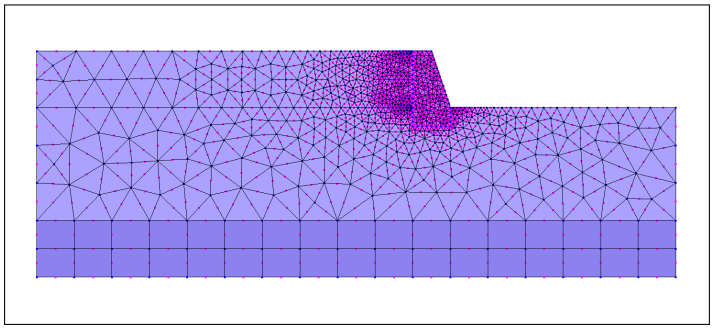
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Maillage : **MAIL**, **GFD_GF**
— Utilitaires : **COMT**, **EXPO**, **IMPR**, **GEN_IIP**, **GEN_IRC**
— Eléments : **MB**, **MB_CP**, **FD**, **FD_FC**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_POI**, **CHAR_PUR**
— Calcul : **TCNL**
— Jeu de données : **lancer**

5.7.47 Test ssnp028

Allure du maillage.

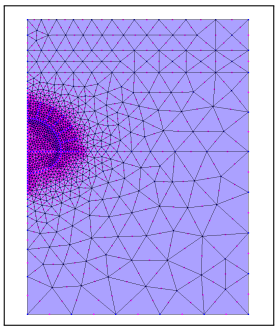


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **GROUPE**, **'MECA_2D_DPLAN'**, **'CONTACT'**
— Modèles :
— Matériaux : **MOHR_COUL_SANS_ECROU**, **ELAS_LINE_ISO**
— Caractéristiques :
— Liaisons : **CONTACT**, **FROT_COUL**
— Conditions limites : **DDL_IMPOSE**

— Chargements : **CONTR_GEOSTAT**, **COUCHE_SOL**, **EXCAV_2D**, **POIDS**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**, **VERIF_CONTACT**, **BASE**
— Résultats : **FICHIER**, **'GMSH'**
— Phasage : **lancer**

5.7.48 Test ssnp029

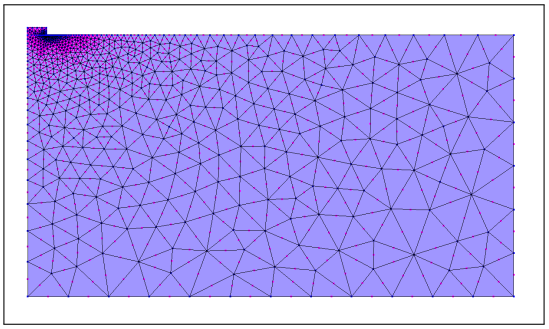
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **GROUPE**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
— Modèles :
— Matériaux : **MOHR_COUL_SANS_ECROU**, **ELAS_LINE_ISO**
— Caractéristiques : **BAR_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **EXCAV_2D**, **POIDS**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**, **BASE**
— Résultats : **FICHIER**, **'GMSH'**
— Phasage : **lancer**

5.7.49 Test ssnp030

Allure du maillage.

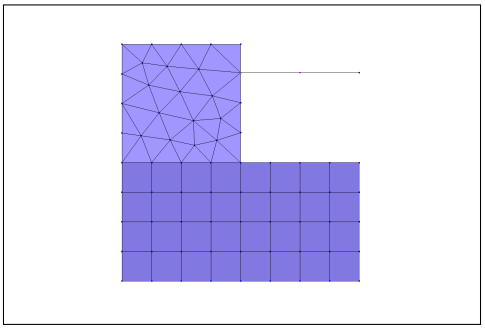


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Domaines : **OPTION**, **POINT**, **LIGNE**, **BOUCLE**, **BLOC**, **mailler**
— Maillages : **FICHIER**, **'GMSH'**, **extraire**, **COOR_2D**

— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
— Matériaux : **ELAS_LINE_ISO**, **VON_MISES_SANS_ECROU**, **MOHR_COUL_SANS_ECROU**
— Caractéristiques :
— Conditions limites : **DDL_IMPOSE**
— Chargements : **POIDS**, **FORC_NOEUD**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**, **extraire**, **DEPLA_2D**
— Etude : **lancer**

5.7.50 Test ssnp031

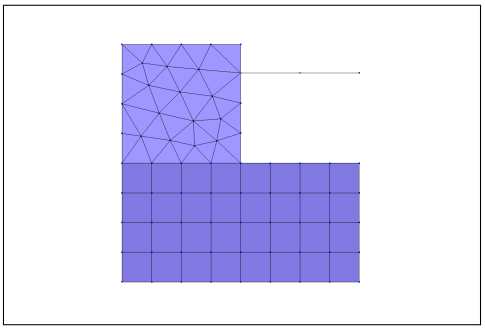
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **MB**, **MB_ELI**, **BB**, **BB_ELI**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_PUR**, **CHAR_DEC**
— Calcul : **MCNL**
— Jeu de données : **lancer**

5.7.51 Test ssnp032

Allure du maillage.

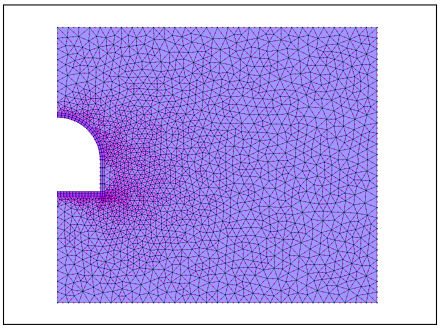


Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **'MECA_2D_DPLAN'**, **'BAR_2D'**
— Matériaux : **ELAS_LINE_ISO**

— Caractéristiques : **BAR_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_ARETE_2D**, **DECONFIN_2D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**
— Résultats : **FICHIER**, **'GMSH'**
— Phasage : **lancer**

5.7.52 Test ssnp033

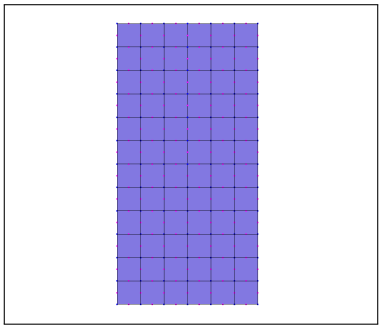
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Domaine : **POINT**, **LIGNE**, **CERCLE**, **BOUCLE**, **SURFACE**, **RECOMBINE**, **OPTION**, **STRUCTURE**, **BLOC**
— Maillages : **FICHIER**, **'GMSH'**, **extraire**, **COOR_2D**
— Modèles : **'MECA_2D_DPLAN'**, **'POUT_2D'**
— Matériaux : **MOHR_COUL_SANS_ECROU**, **ELAS_LINE_ISO**
— Caractéristiques : **POUT_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **CONTR_GEOSTAT**, **COUCHE_SOL**, **EXCAV_2D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**
— Résultats : **FICHIER**, **'GMSH'**, **extraire**, **DEPLA_2D**, **CONTR_2D**, **EFFORT_POUTR_2D**
— Phasage : **lancer**

5.7.53 Test ssnp034

Allure du maillage.

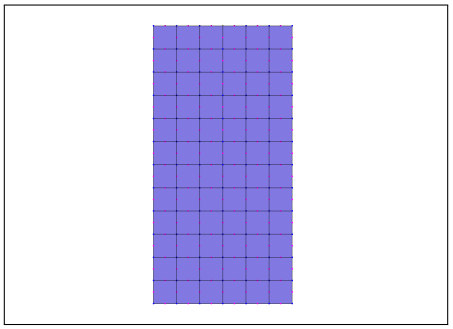


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**
- Eléments : **MB**, **MB_ELI**, **KR**, **KR_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**
- Résultats : **RSV4**, **exporter**

5.7.54 Test ssnp035

Allure du maillage.

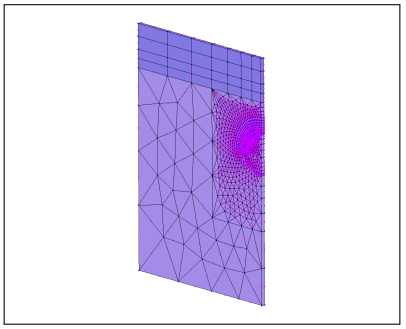


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Domaine : **POINT**, **LIGNE**, **BOUCLE**, **SURFACE**, **RECOMBINE**, **OPTION**, **STRUCTURE**, **BLOC**
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **'MECA_2D_DPLAN'**, **'BAR_2D_FROT'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **BAR_2D**, **BAR_FROT_PLAS**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_NOEUD**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**
- Résultats : **FICHIER**, **'GMSH'**
- Phasage : **lancer**

5.7.55 Test ssnp036

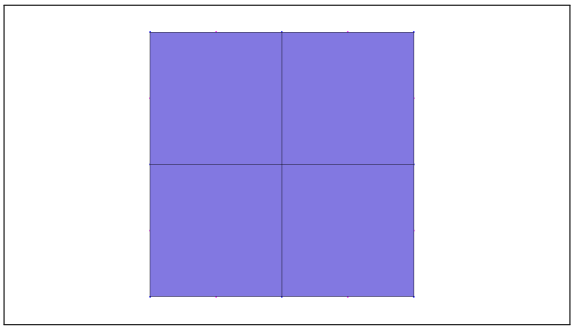
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**
- Eléments : **MT**, **MT_MCSE**, **KR**, **KR_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**, **POI_FV**, **CHAR_LAM**
- Calcul : **MCNL**, **INI**, **STK**, **NDP**, **DTO**, **DPL**
- Jeu de données : **lancer**
- Résultats : **RSV4**, **exporter**

5.7.56 Test ssnp037

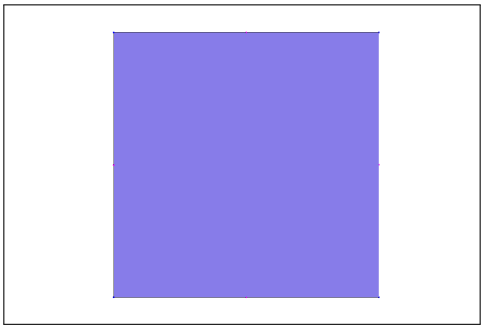
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_BAO**, **CMP_ELAS**, **ELAS_LI**, **CMP_CRT**, **FNC_DPC**, **CMP_POT**, **FNC_DP**, **CMP_ECR**, **ECR_CH**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**
- Chargements : **CHAR_SIG**, **SIG_CST**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.7.57 Test ssnp038

Allure du maillage.

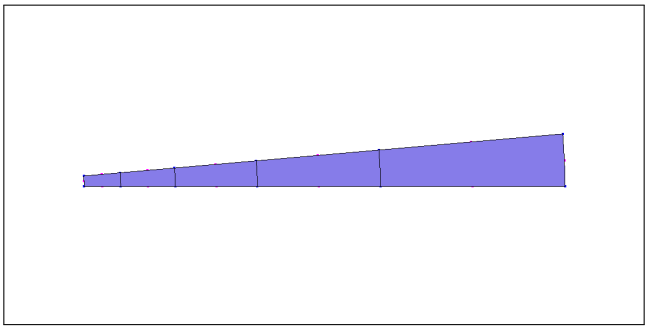


Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_BAO**, **CMP_ELAS**, **ELAS_HSM**, **CMP_CRT**, **FNC_HSM**, **CMP_ECR**, **ECR_HSM**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**

— Chargements : **CHAR_SIG**, **SIG_CST**
— Calcul : **MCNL**
— Jeu de données : **lancer**

5.7.58 Test ssnp039

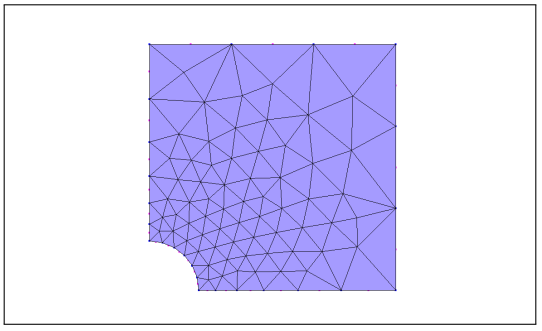
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **MB**, **MB_BAO**, **CMP_ELAS**, **ELAS_LI**, **CMP_CRT**, **FNC_DP**, **CMP_RENF**, **RENF_PP**, **RENF_R**
— Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
— Chargements : **CHAR_SIG**, **SIG_CST**
— Calcul : **MCNL**
— Jeu de données : **lancer**

5.7.59 Test ssnp040

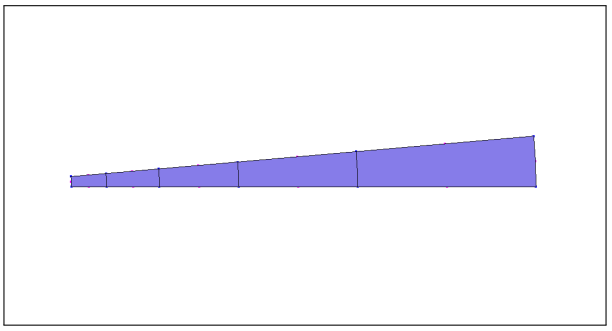
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **MB**, **MB_BAO**, **CMP_ELAS**, **ELAS_ANL**, **CMP_CRT**, **FNC_MC**
— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_SIG**, **SIG_GEO**, **CHAR_PUR**
— Calcul : **MCNL**, **DTO**, **DPL**
— Jeu de données : **lancer**

5.7.60 Test ssnp041

Allure du maillage.

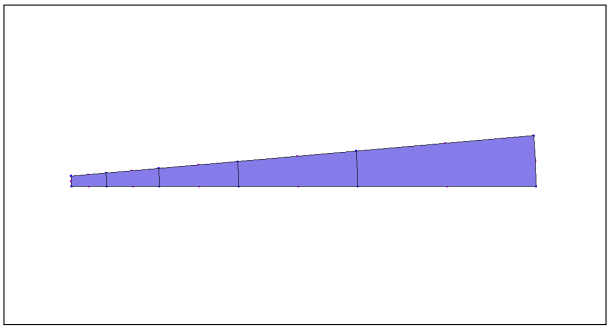


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_MCSE**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**, **FSC**
- Jeu de données : **lancer**

5.7.61 Test ssnp042

Allure du maillage.



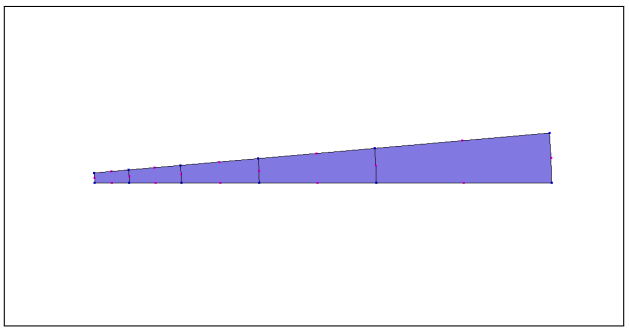
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MB**, **MB_MCSE**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**, **FSR**
- Jeu de données : **lancer**

5.8 Catégorie "structure statique non linéaire surfacique" (ssns)

5.8.1 Test ssns001a

Allure du maillage.

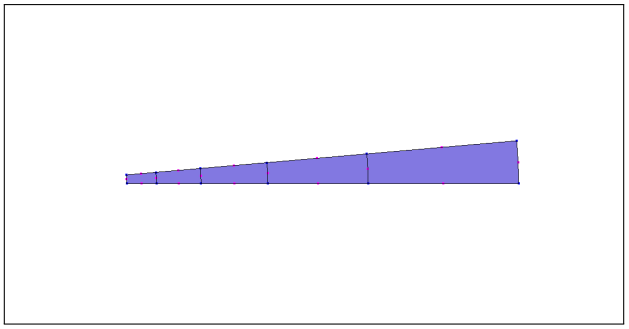


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_MC**, **CO_LC**, **CO_LC_VMSE**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.8.2 Test ssns001b

Allure du maillage.

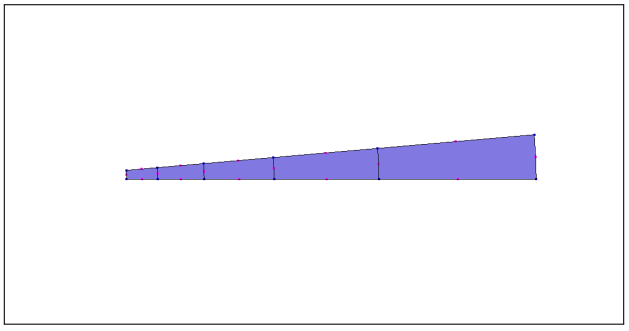


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_MC**, **CO_LC**, **CO_LC_VMAE**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.8.3 Test ssns001c

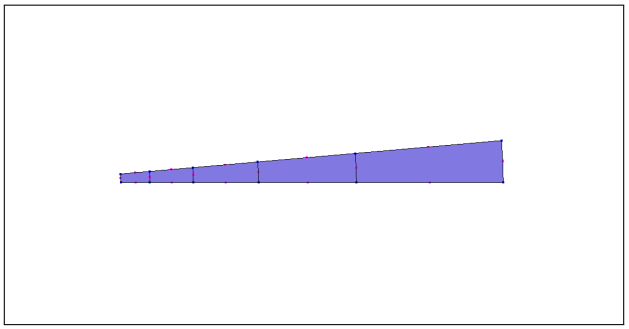
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_MC**, **CO_LC**, **CO_LC_CP**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.8.4 Test ssns001d

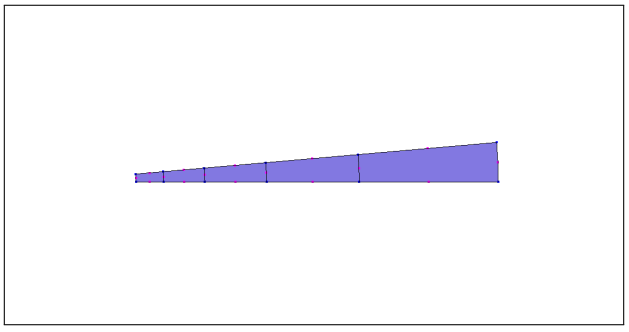
Allure du maillage.



Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_MC**, **CO_LC**, **CO_LC_WWS**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.8.5 Test ssns001e

Allure du maillage.

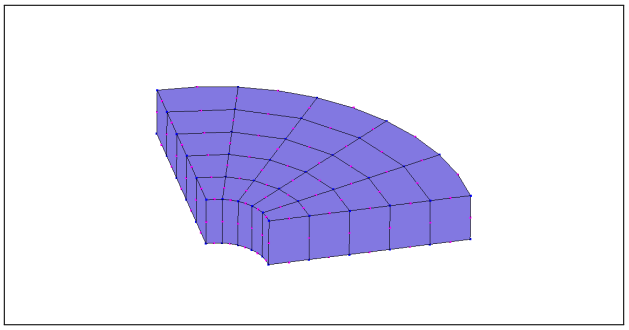


Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Eléments : **CO**, **CO_MC**, **CO_LC**, **CO_LC_WWM**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_SOL**, **GEN_SOL**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.9 Catégorie "structure statique non linéaire volumique" (ssnv)

5.9.1 Test ssnv001a

Allure du maillage.

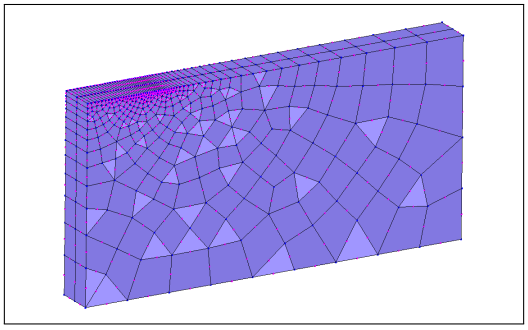


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **DRUCK_PRAG_SANS_ECROU**, **MATER_RENFORCE**, **'BUHAN_SUDRET'**, **ELASTO_PLAST_1D**
- Caractéristiques : **MASSIF_3D RENFOR_3D**, **'RADIAL'**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **PRESS_FACE_3D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.9.2 Test ssnv002a

Allure du maillage.



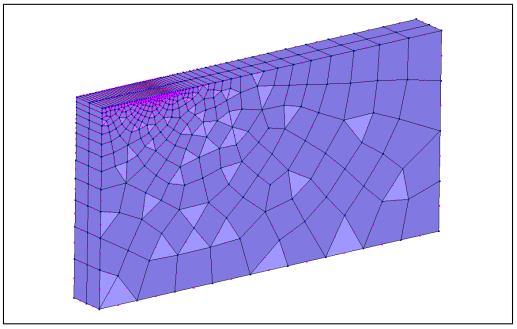
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**, **MOHR_COUL_SANS_ECROU**, **VON_MISES_SANS_ECROU**, **VON_MISES_AVEC**
  DRUCK_PRAG_SANS_ECROU, **DRUCK_PRAG_AVEC_ECROU**, **CRIT_PARABOL**, **CRIT_ORIENT**, **HOEK_BROWN**, **ELAS_DILAT_ISO**, **CAM_CLAY_MODIF**, **PREVOST_HOEG**, **ELAS_LINE_ORTHO**
- Caractéristiques : **MASSIF_3D RENFOR_3D**, **'RADIAL'**

— Conditions limites : **DDL_IMPOSE**
— Chargements : **POIDS**, **FORC_FACE_3D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.9.3 Test ssnv002b

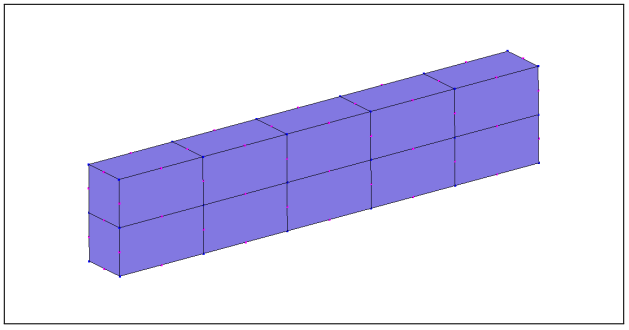
Allure du maillage.



Mots-clés.
— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_MECA**, **'MECA_3D'**
— Matériaux : **MATER_RENFORCE**, **DRUCK_PRAG_SANS_ECROU**, **ELASTO_PLAST_1D**
— Caractéristiques : **MASSIF_3D RENFOR_3D**, **'HOMOGEN'**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **POIDS**, **FORC_FACE_3D**
— Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.9.4 Test ssnv003

Allure du maillage.

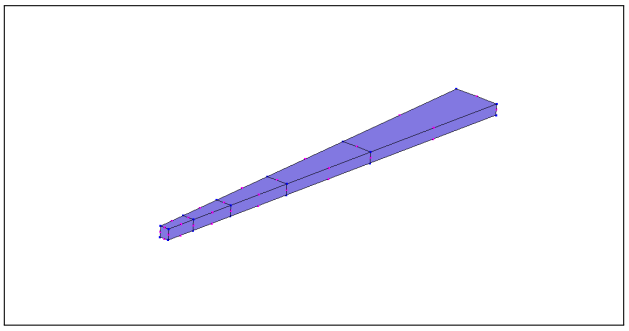


Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **MT**, **MT_ELI**, **FD**, **FD_FC**

— Conditions limites : **COND_NUL**, **GEN_NUL**
— Chargements : **CHAR_POI**, **CHAR_PUR**
— Calcul : **TCNL**
— Jeu de données : **lancer**

5.9.5 Test ssnv004a

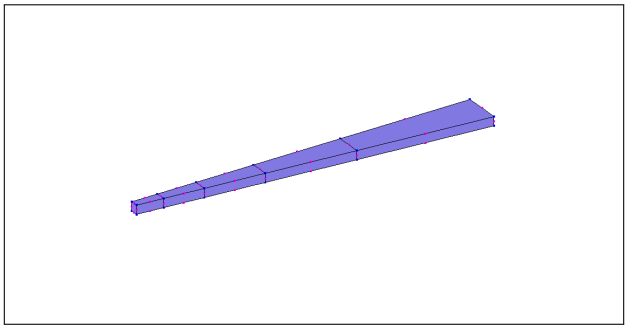
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **MT**, **MT_VE**
— Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
— Chargements : **CHAR_PUR**
— Calcul : **MCNL**
— Jeu de données : **lancer**

5.9.6 Test ssnv004b

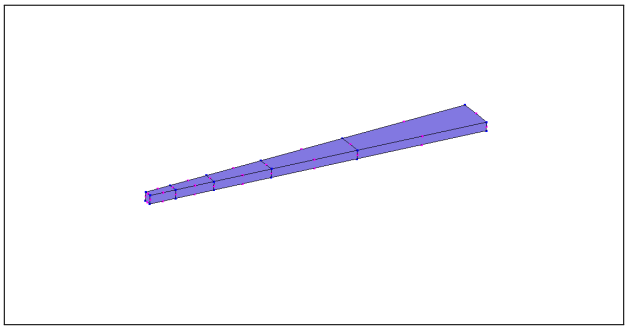
Allure du maillage.



Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Eléments : **MT**, **MT_NO**
— Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
— Chargements : **CHAR_PUR**
— Calcul : **MCNL**
— Jeu de données : **lancer**

5.9.7 Test ssnv004c

Allure du maillage.

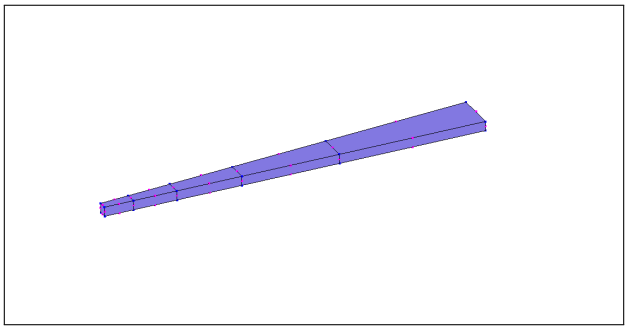


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MT**, **MT_WWS**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.9.8 Test ssnv004d

Allure du maillage.

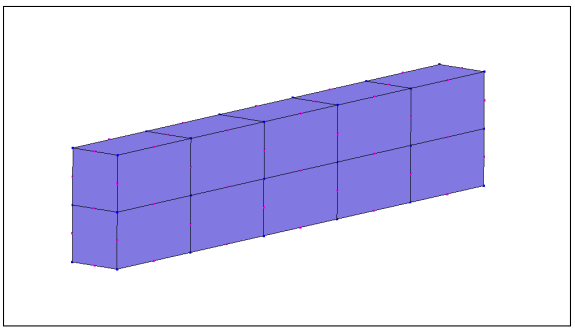


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **MT**, **MT_WWM**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements : **CHAR_PUR**
- Calcul : **MCNL**
- Jeu de données : **lancer**

5.9.9 Test ssnv005a

Allure du maillage.

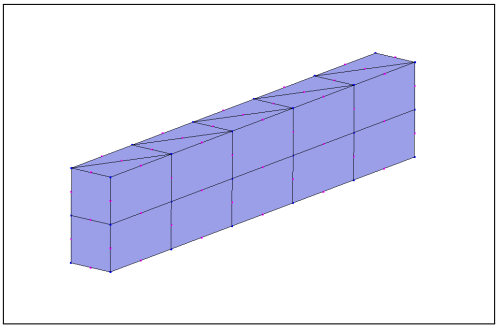


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**, **GFD**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MT**, **MT_CP**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**, **CHAR_PUR**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.9.10 Test ssnv005b

Allure du maillage.

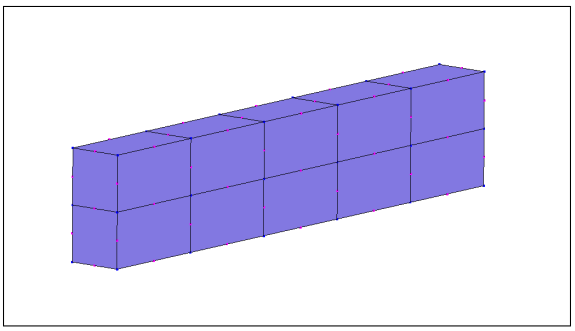


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**, **GFD**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MT**, **MT_CP**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**, **CHAR_PUR**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.9.11 Test ssnv005c

Allure du maillage.

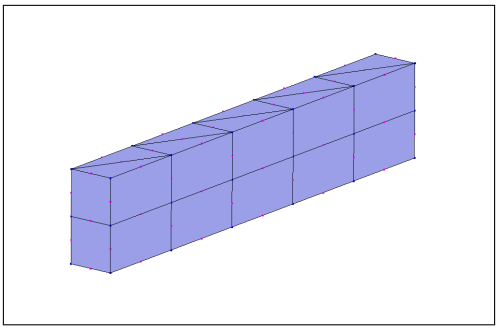


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**, **GFD_GF**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MT**, **MT_CP**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**, **CHAR_PUR**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.9.12 Test ssnv005d

Allure du maillage.



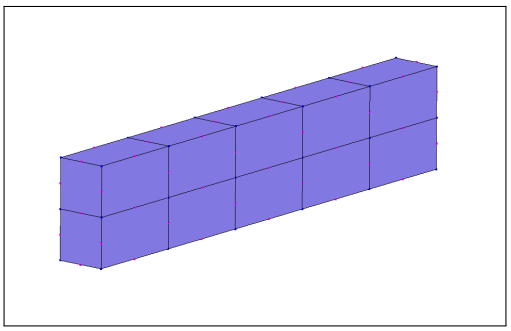
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'GMSH'**, **GFD_GF**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MT**, **MT_CP**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**, **CHAR_PUR**
- Calcul : **TCNL**
- Jeu de données : **lancer**

5.9.13 Test ssnv006

Ce test correspond au test de non regression "bila3_tcnl1" de CESAR.

Allure du maillage.

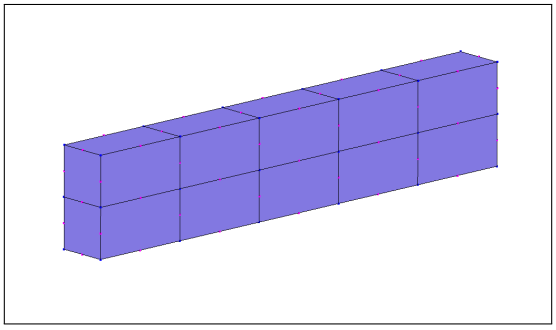


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'CESAR'**
- Utilitaires : **COMT**, **EXPO**
- Eléments : **MT**, **MT_CP**, **FD**, **FD_FC**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_POI**, **CHAR_PUR**
- Calcul : **TCNL**
- Jeu de données : **lancer**
- Résultats : **RSV4**, **formater**

5.9.14 Test ssnv007

Allure du maillage.



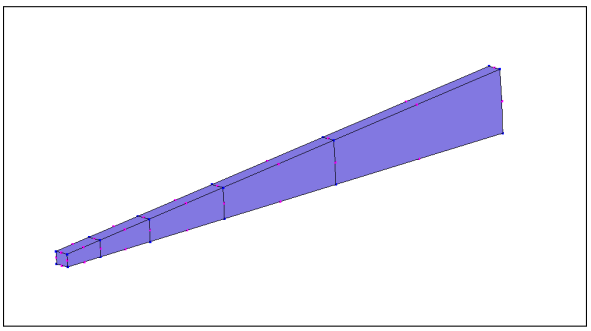
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **CRIT_PARABOL**
- Caractéristiques :
- Liaisons : **CONTACT**, **FROT_COUL**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **POIDS**, **PRESS_FACE_3D**
- Résolutions : **STAT_NON_LINE**, **'MULTI_FRONT'**, **METHOD_ITER**, **'CONTR_INIT'**, **INCREMENTATION**, **FONCT_MULT**, **VERIF_CONTACT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.9.15 Test ssnv008

Ce test correspond au test de non regression "tus3m_mxcp1" de CESAR.

Allure du maillage.



Mots-clés.

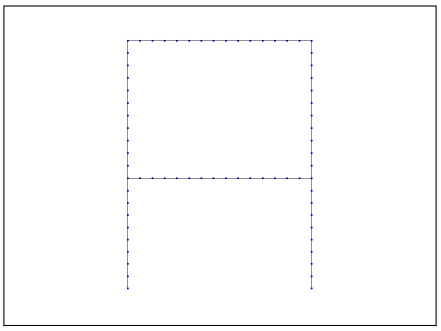
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **MT**, **MT_BJA**, **ET**, **ET_CCH**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
- Chargements :
- Calcul : **MEXO**, **TXO**
- Jeu de données : **lancer**

5.10 Catégorie "structure dynamique linéaire linéique" (sdll)

5.10.1 Test sdll001a

Ce test correspond au test de non regression "port3_mode1" de CESAR.

Allure du maillage.



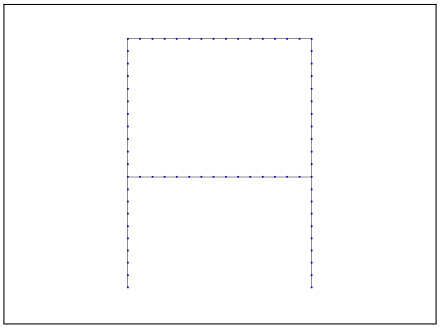
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **EXPO**
- Eléments : **PT**, **PT_STD_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements :
- Calcul : **MODE**
- Jeu de données : **lancer**

5.10.2 Test sdll001b

Ce test correspond au test de non regression "port3_sumo1" de CESAR.

Allure du maillage.



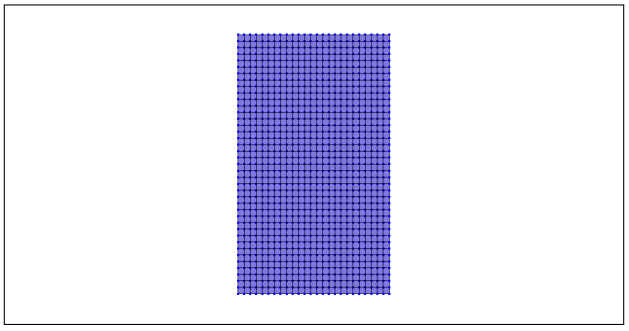
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **PT**, **PT_STD_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **GEN_SOL**
- Calcul : **SUMO**, **CFT**, **SRE**
- Jeu de données : **lancer**

5.11 Catégorie "structure dynamique linéaire surfacique" (sdls)

5.11.1 Test sdls001a

Allure du maillage.

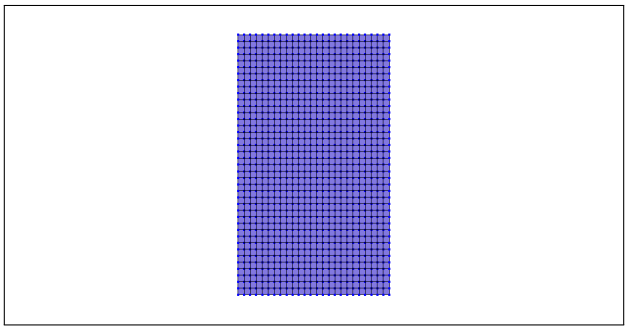


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_HARMO**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.11.2 Test sdls001b

Allure du maillage.

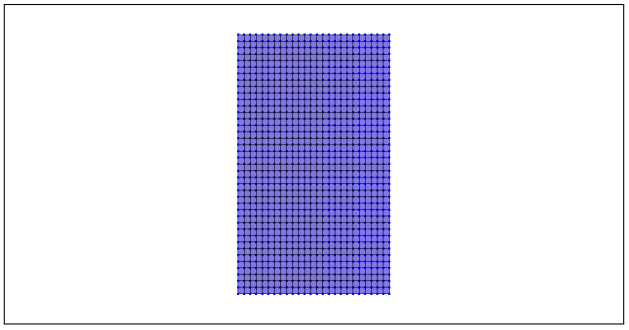


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_HARMO**, **AMORTISSEMENT**, **'RAYLEIGH'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.11.3 Test sdls001c

Allure du maillage.

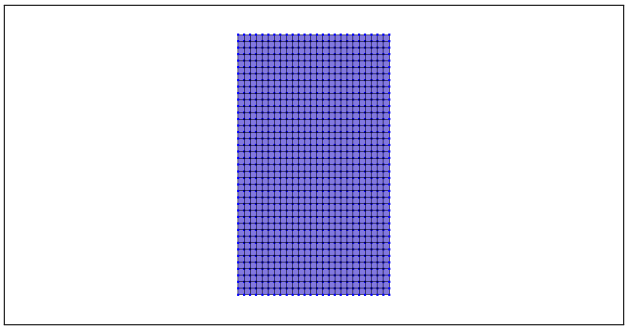


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_TEMP**, **INCREMENTATION**, **AMORTISSEMENT**, **'RAYLEIGH'**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.11.4 Test sdls001d

Allure du maillage.

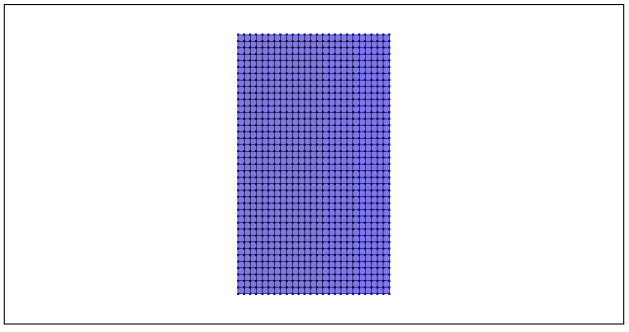


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **MODAL_LINE**, **'VIBRA'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**, **BASE**, **DYNA_LINE_TEMP**, **INCREMENTATION**, **AMORTISSEMENT**, **'MODAL'**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**, **SELECT**, **'TXT'**, **DEPLA_2D**
- Etude : **lancer**

5.11.5 Test sdls002a

Allure du maillage.

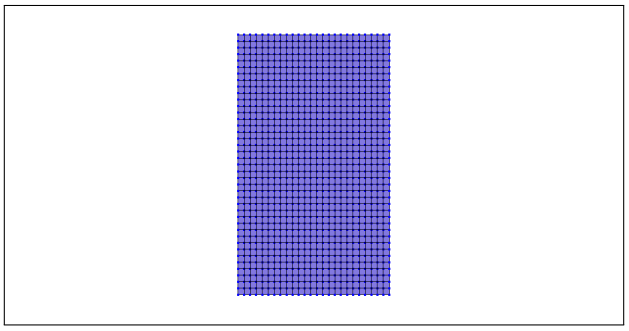


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_HARMO**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.11.6 Test sdls002b

Allure du maillage.

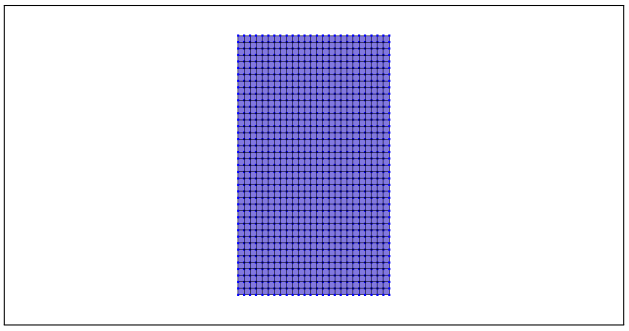


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_HARMO**, **AMORTISSEMENT**, **'RAYLEIGH'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.11.7 Test sdls002c

Allure du maillage.

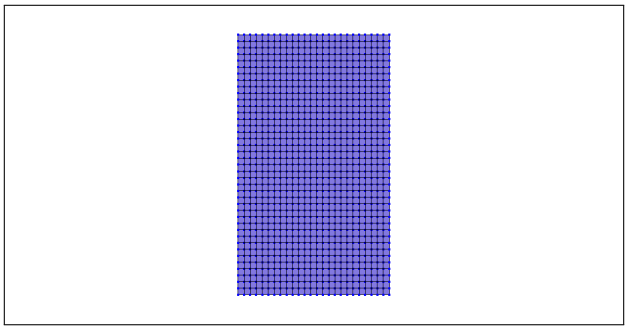


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_TEMP**, **INCREMENTATION**, **AMORTISSEMENT**, **'RAYLEIGH'**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.11.8 Test sdls002d

Allure du maillage.

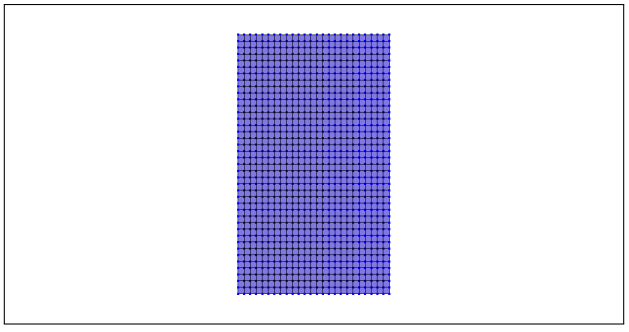


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **MODAL_LINE**, **'VIBRA'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**, **BASE**, **DYNA_LINE_TEMP**, **INCREMENTATION**, **AMORTISSEMENT**, **'MODAL'**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.11.9 Test sdls002e

Allure du maillage.



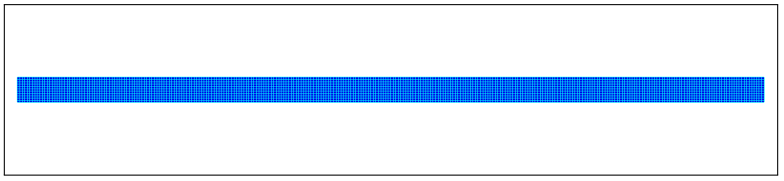
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **MODAL_LINE**, **'VIBRA'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**, **DYNA_LINE_TEMP**, **INCREMENTATION**, **AMORTISSEMENT**, **'MODAL'**, **FONCT_MULT**
- Résultats : **FICHIER**, **'GMSH'**

— Etude : **lancer_aster**

5.11.10 Test sdls003a

Allure du maillage.

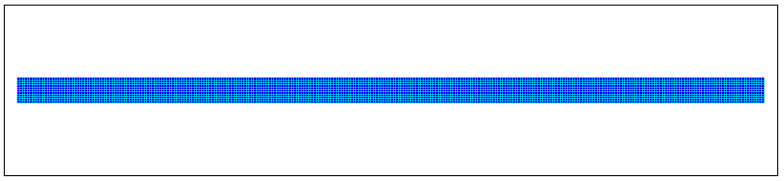


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'MED'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques : **BAR_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_ARETE_2D**, **FORC_NOEUD**
— Résolutions : **DYNA_LINE_TEMP**, **'DIRECTE'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'MED'**, **SELECT**, **'TXT'**, **DEPLA_2D**
— Etude : **lancer**

5.11.11 Test sdls003b

Allure du maillage.

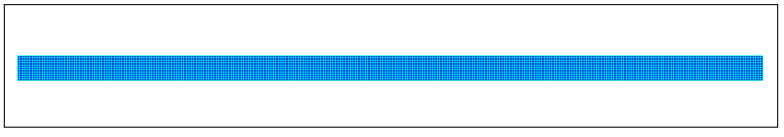


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'MED'**
— Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
— Matériaux : **ELAS_LINE_ISO**
— Caractéristiques : **BAR_2D**
— Conditions limites : **DDL_IMPOSE**
— Chargements : **FORC_ARETE_2D**, **FORC_NOEUD**
— Résolutions : **DYNA_LINE_TEMP**, **'DIRECTE'**, **INCREMENTATION**, **FONCT_MULT**
— Résultats : **FICHIER**, **'MED'**, **SELECT**, **'TXT'**, **DEPLA_2D**
— Etude : **lancer**

5.11.12 Test sdls004a

Allure du maillage.

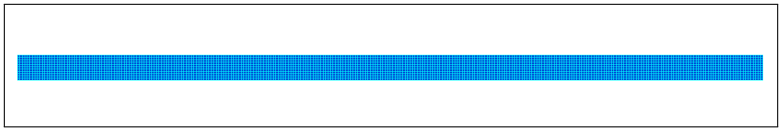


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'MED'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **BAR_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_TEMP**, **'DIRECTE'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FILTRE**, **FICHIER**, **'MED'**, **SELECT**, **'TXT'**, **DEPLA_2D**
- Etude : **lancer**

5.11.13 Test sdls004b

Allure du maillage.



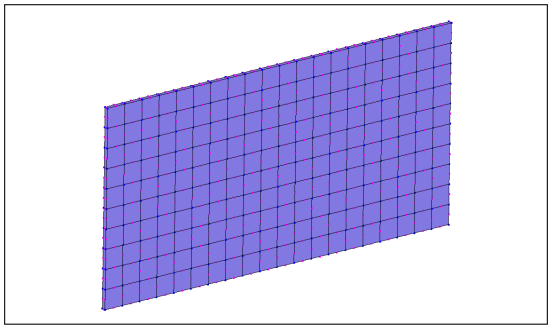
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'MED'**
- Modèles : **FORMUL_MECA**, **'MECA_2D_DPLAN'**, **'BAR_2D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques : **BAR_2D**
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FORC_ARETE_2D**
- Résolutions : **DYNA_LINE_TEMP**, **'DIRECTE'**, **INCREMENTATION**, **FONCT_MULT**
- Résultats : **FILTRE**, **FICHIER**, **'GMSH'**, **SELECT**, **'TXT'**, **DEPLA_2D**
- Etude : **lancer**

5.12 Catégorie "structure dynamique linéaire volumique" (sdlv)

5.12.1 Test sdlv001a

Allure du maillage.

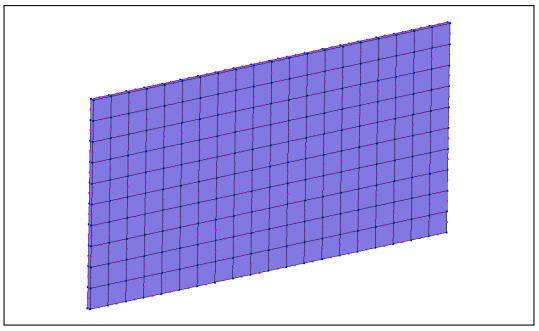


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ELAS_LINE_ISO**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements :
- Résolutions : **MODAL_LINE**, **'VIBRA'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.12.2 Test sdlv002a

Allure du maillage.



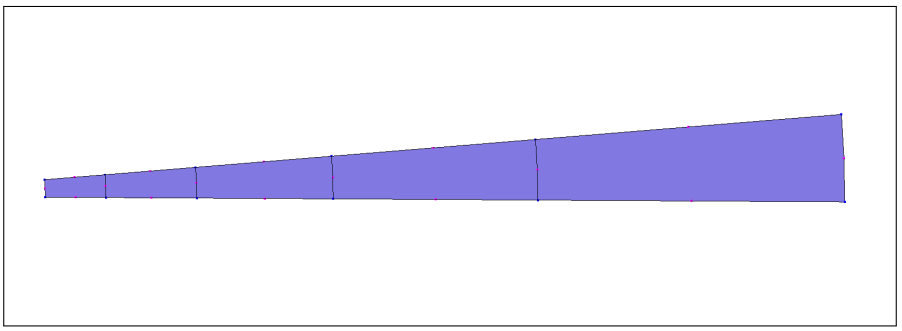
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_MECA**, **'MECA_3D'**
- Matériaux : **ASTER_ELAS**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements :
- Résolutions : **MODAL_LINE**, **'VIBRA'**, **METHOD_VAL_PROP**, **'SOUS_ESPAC'**
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer_aster**

5.13 Catégorie "structure transitoire non linéaire plane" (stnp)

5.13.1 Test stnp001

Allure du maillage.

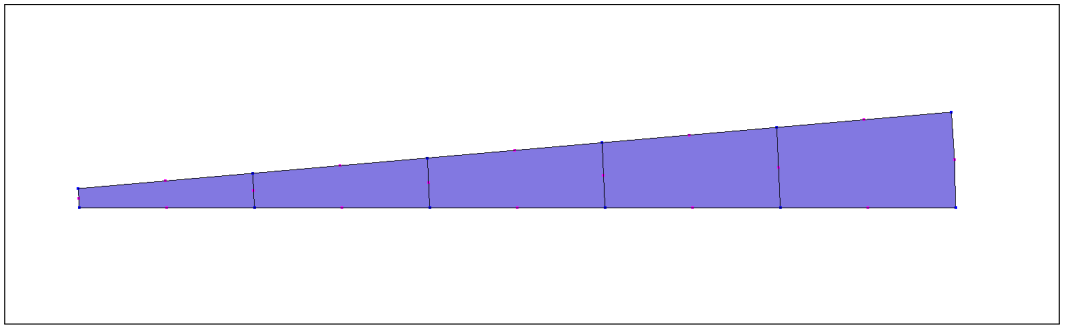


Mots-clés.

— Import Python : modele_cesar (langage CESAR)
— Eléments : **DB**, **DB_CCH**, **EB**, **EB_CCH**
— Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_REP**
— Chargements : **CHAR_ECH**
— Calcul : **TEXO**, **CFT**, **INI**, **QAB**, **PTX**, **SRE**, **MEXO**
— Jeu de données : **lancer**

5.13.2 Test stnp002

Allure du maillage.



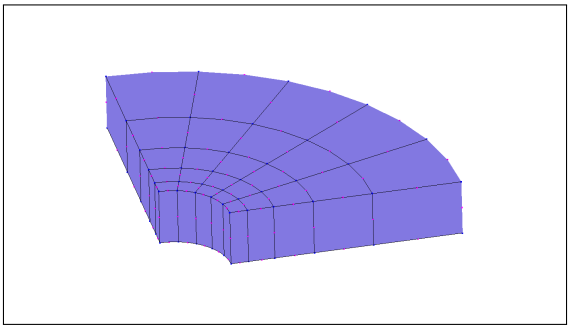
Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Domaine : **POINT**, **LIGNE**, **CERCLE**, **BOUCLE**, **SURFACE**, **RECOMBINE**, **OPTION**, **STRUCTURE**, **BLOC**
— Maillages : **FICHIER**, **'GMSH'**
— Modèles : **FORMUL_DIFFUS**, **'DIFFUS_2D'**, **'ECHANG_2D'**
— Matériaux : **DIFFUS_LINE_2D**, **ECHANG_LINE**, **BETON_JEUNE_AGE**
— Caractéristiques :
— Conditions limites : **DDL_IMPOSE**, **BASE_2D**
— Chargements : **FLUX_ECHANG_LINE_2D**
— Résolutions : **THERM_BETON_JEUNE_AGE**, **ESSAI_QAB**, **FONC_MULT**, **METHOD_ITER**, **MECA_BETON_J**
'MULTI_FRONT'
— Résultats : **FICHIER**, **'GMSH'**
— Etude : **lancer**

5.14 Catégorie "structure transitoire non linéaire volumique" (stnv)

5.14.1 Test stnv001

Allure du maillage.

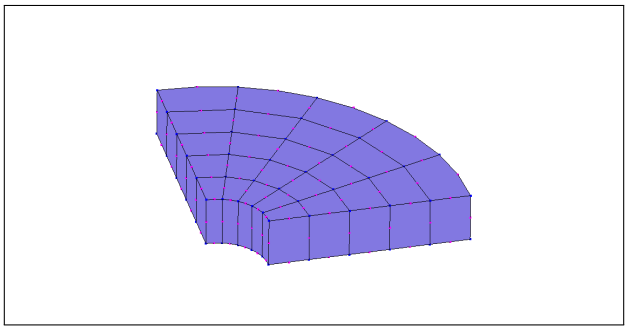


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Eléments : **DT**, **DT_CCH**, **ET**, **ET_CCH**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_ECH**, **CHAR_PUR**
- Calcul : **TEXO**, **CFT**, **INI**, **QAB**, **MEXO**, **TXO**
- Jeu de données : **lancer**

5.14.2 Test stnv002

Allure du maillage.



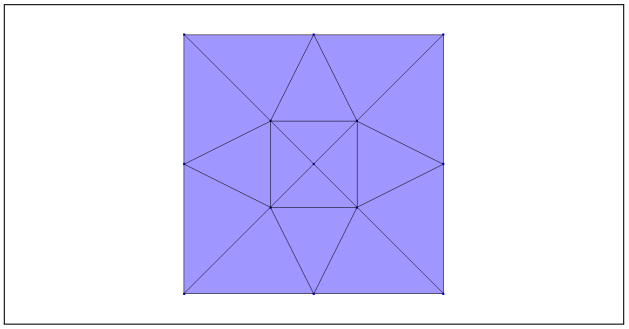
Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Maillages : **FICHIER**, **'GMSH'**
- Modèles : **FORMUL_DIFFUS**, **'DIFFUS_3D'**, **'ECHANG_3D'**
- Matériaux : **DIFFUS_LINE_3D**, **ECHANG_LINE**, **BETON_JEUNE_AGE**
- Caractéristiques :
- Conditions limites : **DDL_IMPOSE**
- Chargements : **FLUX_ECHANG_LINE_3D**, **PRESS_FACE_3D**
- Résolutions : **THERM_BETON_JEUNE_AGE**, **ESSAI_QAB**, **FONC_MULT**, **METHOD_ITER**, **MECA_BETON_J**
  'MULTI_FRONT'
- Résultats : **FICHIER**, **'GMSH'**
- Etude : **lancer**

5.15 Catégorie "thermique permanent linéaire plane" (tplp)

5.15.1 Test tplp001

Allure du maillage.



Mots-clés.

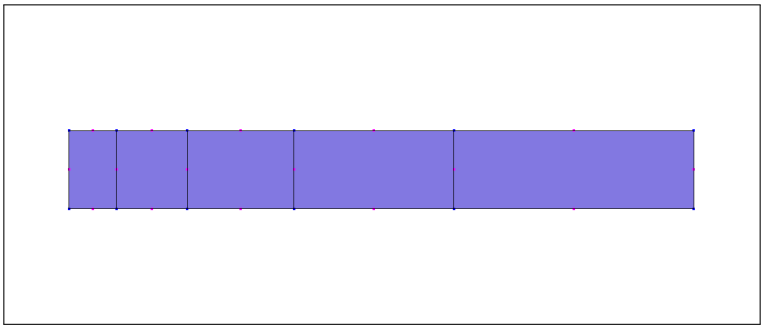
- Import Python : modele_cesar (langage CESAR)
- Eléments : **DB**, **DB_CCH**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements : **CHAR_DVU**
- Calcul : **LINE**
- Jeu de données : **lancer**

5.16 Catégorie "thermique transitoire linéaire plane" (ttlp)

5.16.1 Test ttlp001

Ce test correspond au test de non regression "tubxt_dtli1" de CESAR.

Allure du maillage.



Mots-clés.

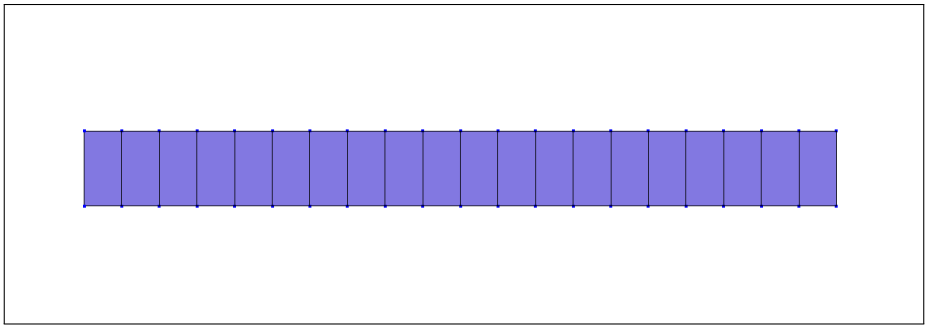
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **DB**, **DB_CCH**
- Conditions limites : **COND_IMP**, **GEN_IMP**
- Chargements :
- Calcul : **DTLI**, **INI**, **LIM**, **KVCOND**, **SRE**
- Jeu de données : **lancer**

5.17 Catégorie "thermique transitoire non linéaire plane" (ttnp)

5.17.1 Test ttnp001

Ce test correspond au test de non regression "ech2d_dtnl1" de CESAR.

Allure du maillage.



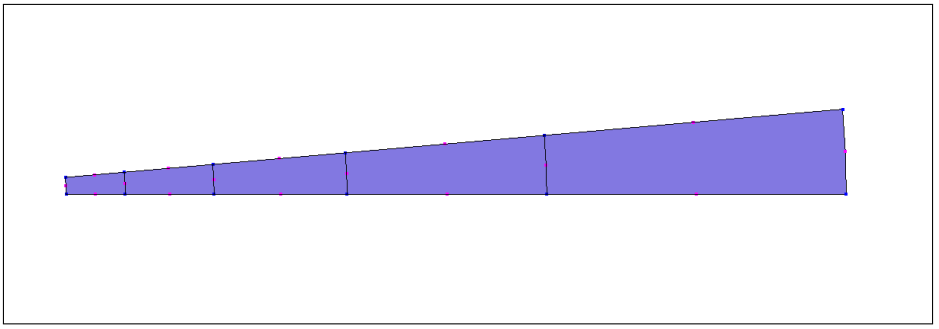
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **DB**, **DB_CCH**, **EB**, **EB_NLG**
- Conditions limites :
- Chargements :
- Calcul : **DTNL**, **ENL**, **SRE**
- Jeu de données : **lancer**

5.17.2 Test ttnp002

Ce test correspond au test de non regression "tus2t_ptx2" de CESAR.

Allure du maillage.



Mots-clés.

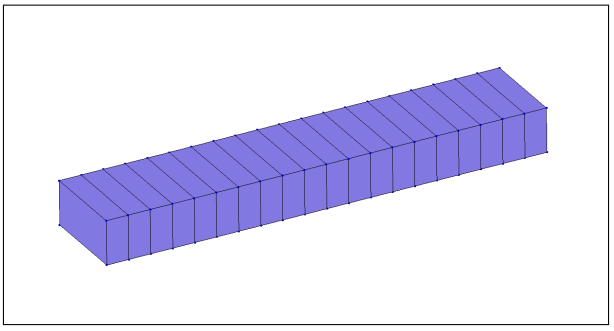
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**, **GEFI**
- Eléments : **DB**, **DB_CCH**, **EB**, **EB_CCH**
- Conditions limites :
- Chargements : **CHAR_ECH**
- Calcul : **TEXO**, **CFT**, **INI**, **QAB**, **PTX**, **SRE**
- Jeu de données : **lancer**

5.18 Catégorie "thermique transitoire non linéaire volumique" (ttnv)

5.18.1 Test ttnv001

Ce test correspond au test de non regression "ech3d_dtnl1" de CESAR.

Allure du maillage.



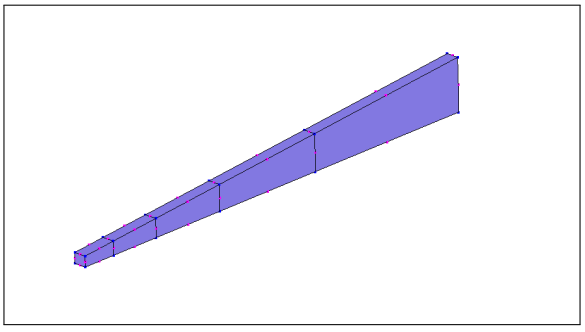
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **DT**, **DT_CCH**, **ET**, **ET_NLG**
- Conditions limites :
- Chargements :
- Calcul : **DTNL**, **ENL**, **SRE**
- Jeu de données : **lancer**

5.18.2 Test ttnv002

Ce test correspond au test de non regression "tus3t_texo1" de CESAR.

Allure du maillage.



Mots-clés.

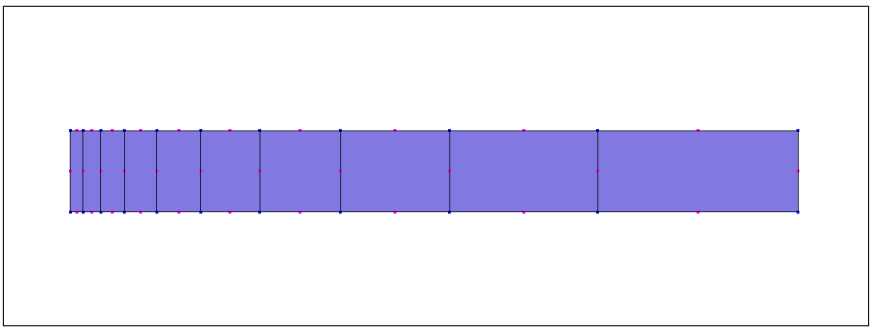
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **DT**, **DT_CCH**, **ET**, **ET_CCH**
- Conditions limites :
- Chargements : **CHAR_ECH**
- Calcul : **TEXO**, **CFT**, **INI**, **QAB**
- Jeu de données : **lancer**

5.19 Catégorie "poreux transitoire linéaire plan" (ptlp)

5.19.1 Test ptlp001

Ce test correspond au test de non regression "puiax_mpli1" de CESAR.

Allure du maillage.



Mots-clés.

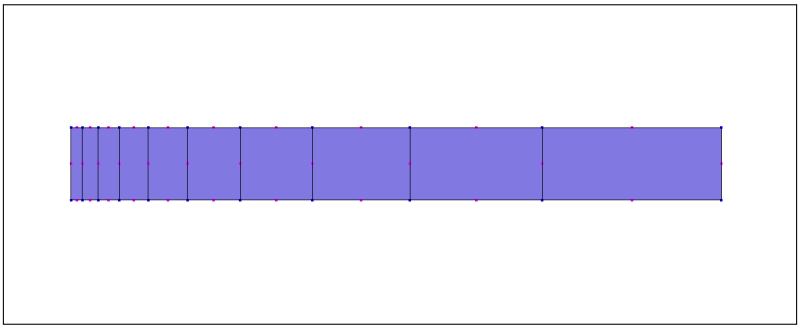
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**, **GEN_IIP**, **GEN_IRC**
- Eléments : **OB**, **OB_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**
- Chargements : **CHAR_PUR**
- Calcul : **MPLI**, **LIM**, **KVCOND**, **CFT**, **SRE**
- Jeu de données : **lancer**

5.20 Catégorie "poreux transitoire non linéaire plan" (ptnp)

5.20.1 Test ptnp001

Ce test correspond au test de non regression "puiax_mpnl1" de CESAR.

Allure du maillage.



Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**, **GEN_IIP**, **GEN_IRC**
- Eléments : **OB**, **OB_DPSE**, **OB_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**
- Chargements : **CHAR_PUR**

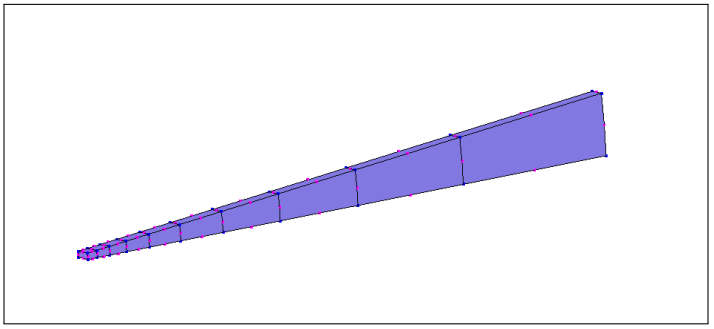
— Calcul : **MPNL**, **LIM**, **KVCOND**, **CFT**, **SRE**
— Jeu de données : **lancer**

5.21 Catégorie "poreux transitoire linéaire volumique" (ptlv)

5.21.1 Test ptlv001

Ce test correspond au test de non regression "tu5ot_mpli1" de CESAR.

Allure du maillage.



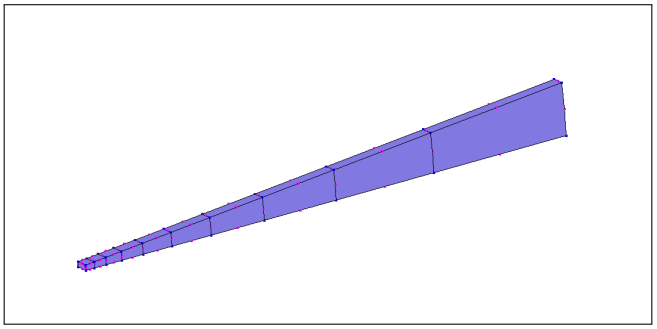
Mots-clés.
— Import Python : modele_cesar (langage CESAR)
— Utilitaires : **COMT**, **IMPR**
— Eléments : **OT**, **OT_ELI**
— Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**, **COND_REP**
— Chargements : **CHAR_PUR**
— Calcul : **MPLI**, **LIM**, **KVCOND**, **CFT**, **SRE**
— Jeu de données : **lancer**

5.22 Catégorie "poreux transitoire non linéaire volumique" (ptnv)

5.22.1 Test ptnv001

Ce test correspond au test de non regression "tu5ot_mpnl1" de CESAR.

Allure du maillage.



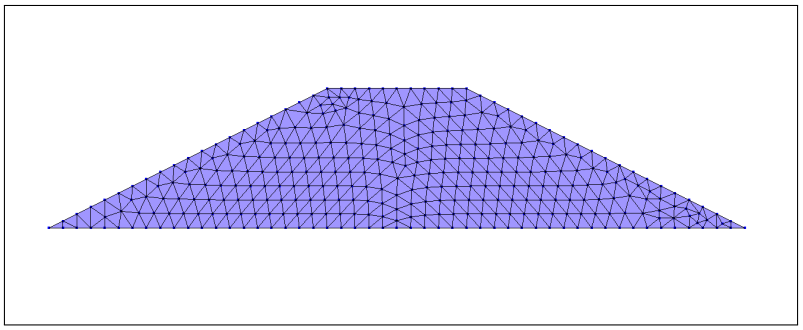
Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **OT**, **OT_DPSE**, **OT_ELI**
- Conditions limites : **COND_NUL**, **GEN_NUL**, **COND_IMP**, **GEN_IMP**, **COND_REP**
- Chargements : **CHAR_PUR**
- Calcul : **MPNL**, **LIM**, **KVCOND**, **CFT**, **SRE**
- Jeu de données : **lancer**

5.23 Catégorie "hydrogéologie permanente non linéaire plane" (hpnp)

5.23.1 Test hpnp001

Ce test correspond au test de non regression "digue_surf1" de CESAR.

Allure du maillage.



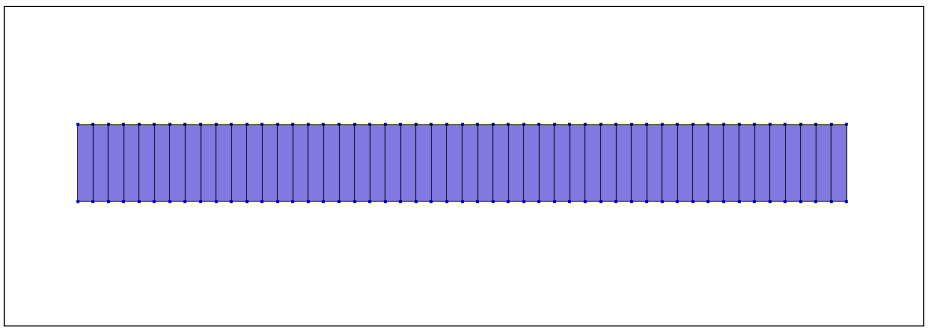
Mots-clés.
- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**, **GEN_IIP**, **GEN_IRC**
- Eléments : **SB**
- Conditions limites : **COND_IMP**, **GEN_IMP**
- Chargements :
- Calcul : **SURF**, **SUI**
- Jeu de données : **lancer**

5.24 Catégorie "hydrogéologie transitoire non linéaire plane" (htnp)

5.24.1 Test htnp001

Ce test correspond au test de non regression "dtnl2_mulns" de CESAR.

Allure du maillage.



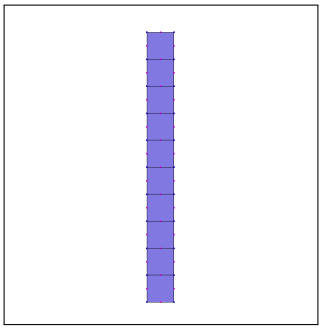
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**
- Eléments : **DB**, **DB_NLG**
- Conditions limites : **COND_NUL**, **GEN_NUL**
- Chargements :
- Calcul : **DTNL**, **INI**, **SRE**, **MUL**
- Jeu de données : **lancer**

5.24.2 Test htnp002

Ce test correspond au test de non regression "colo1_nsat1" de CESAR.

Allure du maillage.



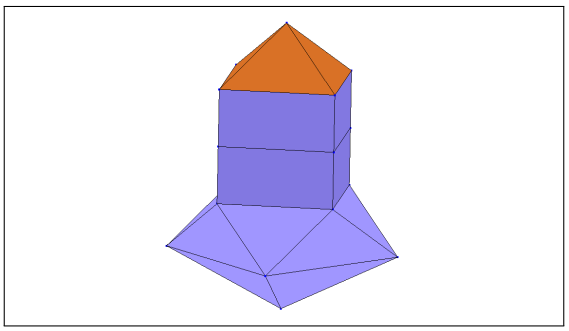
Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Utilitaires : **COMT**, **IMPR**
- Eléments : **DB**, **DB_CP**
- Conditions limites : **COND_IMP**, **GEN_IMP**
- Chargements : **CHAR_FUR**
- Calcul : **NSAT**, **CFT**, **INI**, **LIM**, **KVCOND**, **SRE**
- Jeu de données : **lancer**

5.25 Catégorie "fonctionnalités diverses" (zzzz)

5.25.1 Test zzzz001a

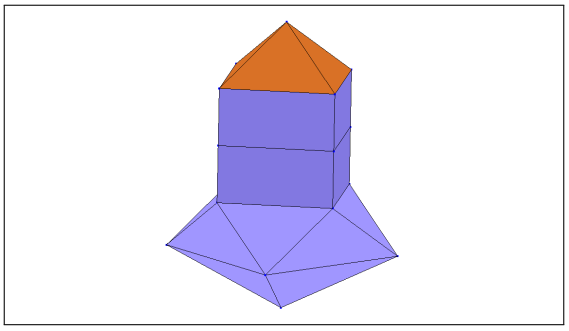
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote), libMEDMEM_Swig (langage MED Mémoire)
- Maillages : **MESHING**, **FICHIER**, **'GMSH'**, **exporter**

5.25.2 Test zzzz001b

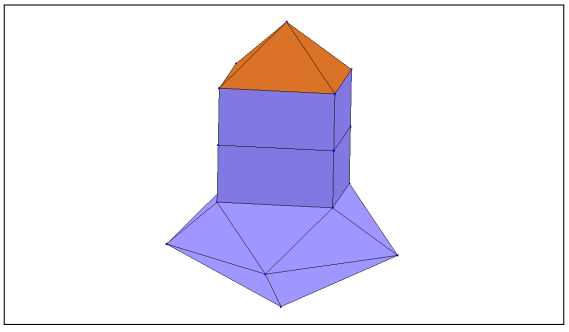
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote), libMEDMEM_Swig (langage MED Mémoire)
- Maillages : **MESHING**, **FICHIER**, **'ASTER'**, **exporter**

5.25.3 Test zzzz001c

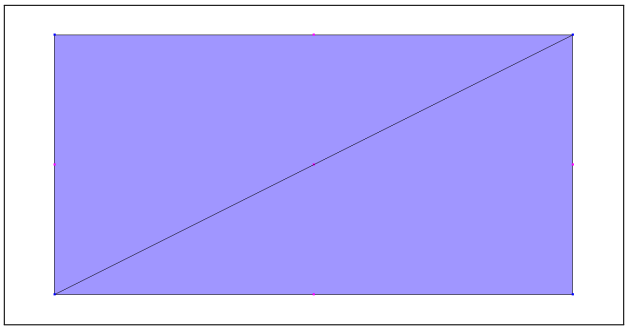
Allure du maillage.



Mots-clés.
- Import Python : modele_donnees (langage Pilote), libMEDMEM_Swig (langage MED Mémoire)
- Maillages : **MESHING**, **FICHIER**, **'MED'**, **exporter**

5.25.4 Test zzzz002

Allure du maillage.

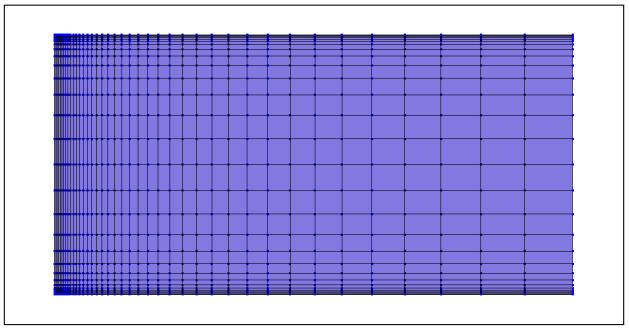


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **'ASTER'**, **convertir**

5.25.5 Test zzzz003

Allure du maillage.

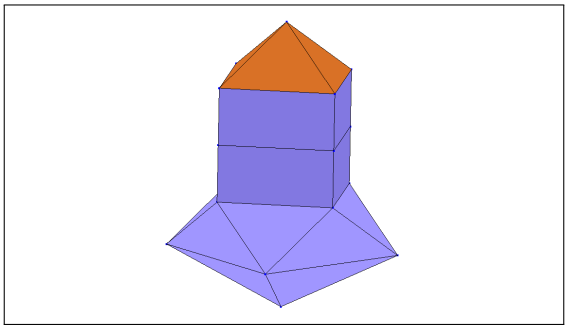


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **'ASTER'**, **convertir**

5.25.6 Test zzzz004a

Allure du maillage.

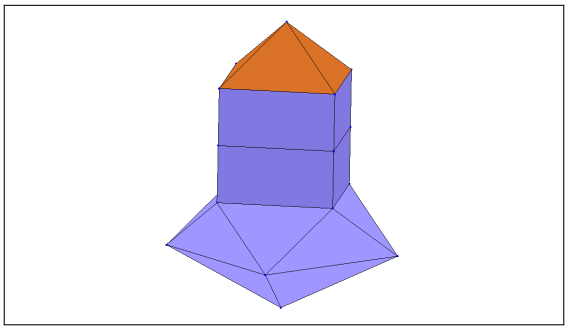


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **'MED'**, **convertir**

5.25.7 Test zzzz004b

Allure du maillage.

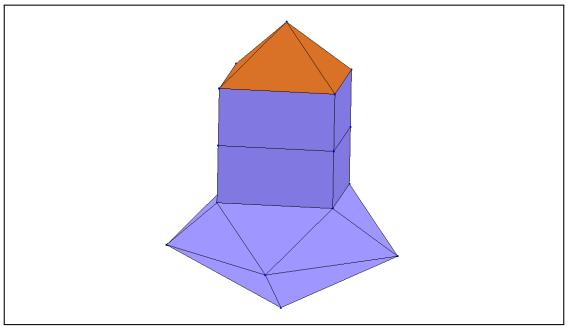


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'MED'**, **'GMSH'**, **convertir**

5.25.8 Test zzzz004c

Allure du maillage.

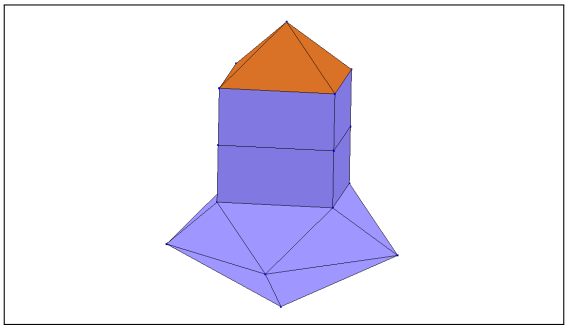


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **'MED'**, **convertir**

5.25.9 Test zzzz004d

Allure du maillage.

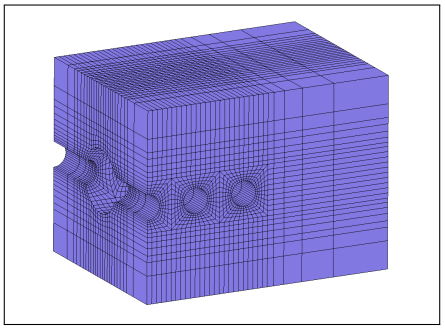


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'MED'**, **'GMSH'**, **convertir**

5.25.10 Test zzzz005a

Allure du maillage.

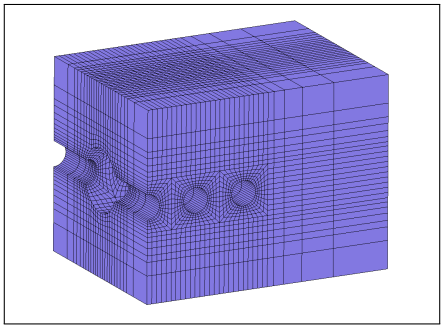


Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'GMSH'**, **'MED'**, **convertir**

5.25.11 Test zzzz005b

Allure du maillage.

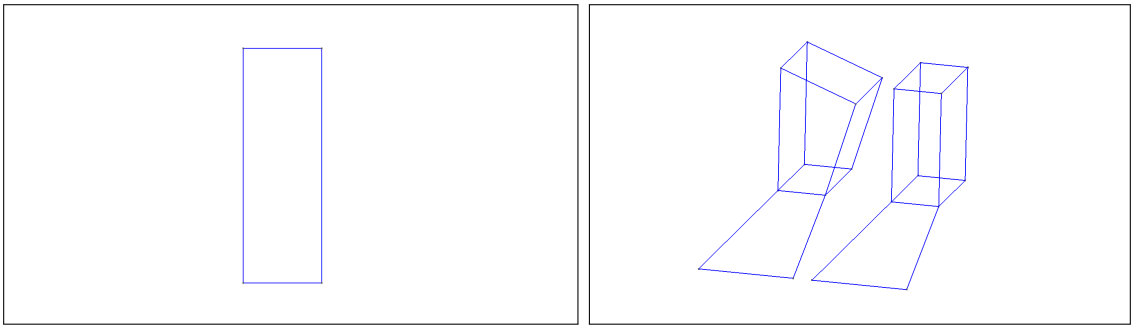


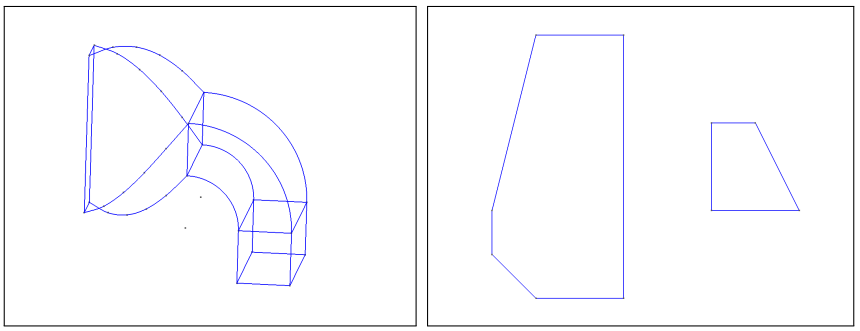
Mots-clés.

— Import Python : modele_donnees (langage Pilote)
— Maillages : **FICHIER**, **'MED'**, **'GMSH'**, **convertir**

5.25.12 Test zzzz006

Allure du domaine.





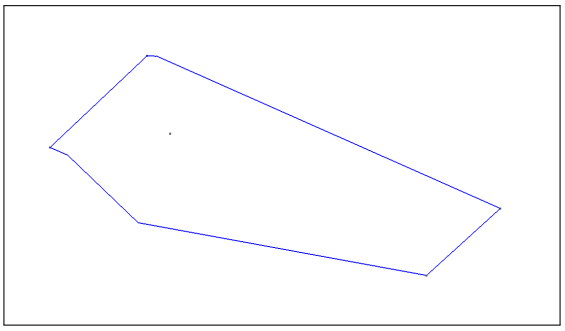
Mots-clés.

— Import Python : modele_donnees (langage Pilote)

— Domaine : **POINT**, **LIGNE**, **SURFACE**, **VOLUME**, **BOUCLE**, **BLOC**, **EXTRUSION**, **TRANSFORMATION**, **STRUCTURE**, **RECOMBINE**, **FINESSE**, **DESTRUCTION**, **OPTION**, **ENTITES**, **add**, **ID**, **mailler**

5.25.13 Test zzzz007

Allure du domaine.



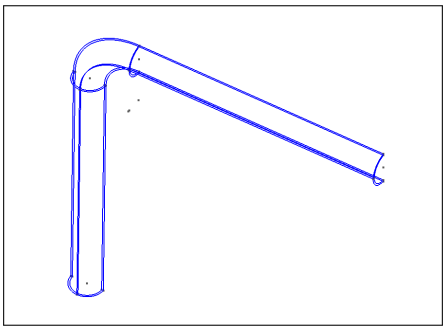
Mots-clés.

— Import Python : modele_donnees (langage Pilote)

— Domaine : **POINT**, **LIGNE**, **SPLINE**, **BOUCLE**, **SURFACE**, **ASSEMBLAGE**, **mailler**

5.25.14 Test zzzz008

Allure du domaine.



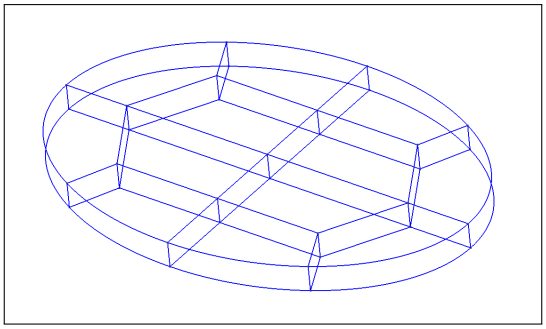
Mots-clés.

— Import Python : modele_donnees (langage Pilote)

— Domaine : **POINT**, **CERCLE**, **LIGNE**, **BOUCLE**, **SURFACE**, **RECOMBINE**, **EXTRUSION**, **BLOC**, **mailler**

5.25.15 Test zzzz009

Allure du domaine.

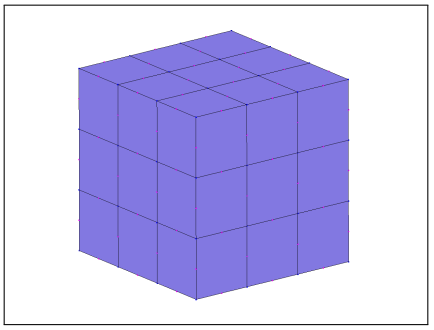


Mots-clés.

- Import Python : modele_donnees (langage Pilote)
- Domaine : **POINT**, **ELLIPSE**, **LIGNE**, **BOUCLE**, **SURFACE**, **STRUCTURE**, **RECOMBINE**, **EXTRUSION**, **BLOC**, **mailler**

5.25.16 Test zzzz010

Allure du domaine.

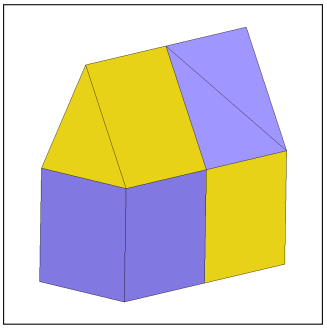


Mots-clés.

- Import Python : modele_cesar (langage CESAR)
- Maillage : **MAIL**, **'CESAR'**
- Résultats : **RSV4**, **exporter**

5.25.17 Test zzzz011

Allure du domaine.



Mots-clés.

— Import Python : modele_cesar (langage CESAR)
— Utilitaires : **EXPO**

Index des classes du langage Pilote

A

ACCEL_2D . . . 89
ACCEL_3D . . . 89
ACTIVATION . . . 56
ADHER . . . 60
AFFECT . . . 8
AMORTISSEMENT . . . 79
ANALYSE . . . 74
ASSEMBLAGE . . . 17
ASTER_ELAS . . . 41
ASTER_ELAS_ORTH . . . 42
ASTER_VMIS_ISOT_LINE . . . 42
AXISYM . . . 48

B

BAR_2D . . . 51
BAR_3D . . . 51
BAR_FROT_BILINE . . . 52
BAR_FROT_LINE . . . 52
BAR_FROT_PLAS . . . 52
BAR_FROT_RACINE . . . 53
BASE . . . 83
BASE_2D . . . 63
BASE_3D . . . 63
BETON_JEUNE_AGE . . . 34
BLOC . . . 17
BOUCLE . . . 16
BSPLINE . . . 16

C

CAM_CLAY_MODIF . . . 38
CARACTERISTIQUE . . . 46
CAS_CHARGEMENT . . . 64
CERCLE . . . 15
CHARGE . . . 65
CHARGE_ARETE . . . 65
CHARGE_ELEMENT . . . 65
CHARGE_FACE . . . 65
CHARGE_NOEUD . . . 65
CHARGEMENT . . . 64
CISAIL_ARETE_2D . . . 67
COHERENCE . . . 20
COMPORTEMENT . . . 33
COMPOSITE . . . 44
COND_LIMITE . . . 62
CONTACT . . . 60
CONTR_2D . . . 90
CONTR_3D . . . 90
CONTR_GEOSTAT . . . 73
CONTR_PLANE . . . 47
CONTR_UNIF . . . 72
COOR_2D . . . 29
COOR_3D . . . 30
COQUE . . . 54
COQUE_MULTI . . . 56
COUCHE . . . 56
COUCHE_SOL . . . 73
CRIT_ORIENT . . . 39
CRIT_PARABOL . . . 37

D

DDL_IMPOSE . . . 63
DECONFIN_2D . . . 68
DECONFIN_3D . . . 69
DEFOR_PLANE . . . 48
DEFORM_TOTAL_2D . . . 91
DEFORM_TOTAL_3D . . . 91
DEPLA_2D . . . 87
DEPLA_2D_ROTA . . . 88
DEPLA_3D . . . 88
DEPLA_3D_ROTA . . . 89
DESTRUCTION . . . 21
DIFFUS_LINE_2D . . . 45
DIFFUS_LINE_3D . . . 45
DOMAINE . . . 12
DRUCK_PRAG_AVEC_ECROU . . . 36
DRUCK_PRAG_SANS_ECROU . . . 36
DYNA_LINE_HARMO . . . 79
DYNA_LINE_TEMP . . . 78

E

ECHANG_LINE . . . 46

ECRO_LINE 43
EFFORT_BAR_2D 93
EFFORT_BAR_3D 93
EFFORT_POUTR_2D 93
EFFORT_POUTR_3D 93
ELAS 43
ELAS_DILAT_ISO 41
ELAS_LINE_ISO 33
ELAS_LINE_ORTHO 34
ELASTO_PLAST_1D 44
ELLIPSE 15
ENTITES 21
ESSAI_QAB 80
ESTIM_ERREUR 82
ETUDE 10
EXCAV_2D 68
EXCAV_3D 68
EXTRACT 10
EXTRUSION 18

F

FIBRE 55
FICHIER 7
FIELD_ 95
FILTRE 84
FINESSE 20
FLUX_ECHANG_LINE_2D 67
FLUX_ECHANG_LINE_3D 67
FONCT_MULT 76
FORC_ARETE_2D 66
FORC_FACE_3D 70
FORC_MAILLE_COQ 71
FORC_MAILLE_POUT_2D 71
FORC_MAILLE_POUT_3D 72
FORC_NOEU 66
FORMUL_DIFFUS 32
FORMUL_MECA 31
FROT_COUL 61

G

GEOMETRIE 47
GLISS 61
GMESH 27
GRAD_TEMP_2D 94
GRAD_TEMP_3D 95
GRANDEUR 10, 86
GROUP_COUCHE 56
GROUP_FIBRE 55
GROUPE 28

H

HOEK_BROWN 40

I

ID 22
INCREMENTATION 76

J

JOINT 61

L

LIAISON 57
LIEU 9
LIGNE 14

M

MAILLAGE 23
MASSIF_3D 49
MATER_RENFORCE 43
MATERIAU 32
MATR_ELEM 59
MECA_BETON_JEUNE_AGE 81
METHOD_ITER 75
METHOD_VAL_PROP 78
MODAL_LINE 77
MODELE 30
MOHR_COUL_SANS_ECROU 35

O

OPERATION 13
OPTION 21

P

PARAM_EXEC 11
PHASAGE 12
PHASE 83
POIDS 71
POINT 14
POST_TRAITEMENT 81
POUT_2D 53
POUT_3D 54
POUT_3D_MULTI 55
PRESS_ARETE_2D 66
PRESS_FACE_3D 70
PRESS_HYDRO_2D 67
PRESS_HYDRO_3D 70
PREVOST_HOEG 39
PROPRIETE 8

R

REAC_2D 92
REAC_3D 92

RECOMBINE 20
RELA_LINE 58
RENFOR_3D 50
RENFOR_AXISYM 49
RENFOR_DEFOR_PLANE 48
RESOLUTION 82
RESULTAT 84

S

SELECT 84
SPLINE 15
STAT_LINE 74
STAT_NON_LINE 74
STRUCTURE 19
SURFACE 16

T

TEMP 94
THERM_BETON_JEUNE_AGE 80
TRANSFORMATION 18
TRESCA_ANISO 40

V

VERIF_CONTACT 77
VERMEER 37
VOLUME 17
VON_MISES_AVEC_ECROU 36
VON_MISES_SANS_ECROU 35

Index des classes du langage CESAR

A

AMO 236
AX 175
AX_CHAR 221
AX_ELI 175
AXIF 220

B

BB 170
BB_ELI 170
BT 170
BT_ELI 171

C

CFT 237
CHAR 207
CHAR_CNU 208
CHAR_CUR 208
CHAR_DEC 213
CHAR_DVU 208
CHAR_ECH 209
CHAR_EFD 209
CHAR_FNU 210
CHAR_FOS 211
CHAR_FUR 211
CHAR_LAM 211
CHAR_PHS 214
CHAR_PNC 215
CHAR_PNU 214
CHAR_POI 216
CHAR_PUR 216
CHAR_SIG 217
CHAR_SOL 219
CMA 238
CMP_CRT 132
CMP_ECR 136
CMP_ELAS 130
CMP_POT 135
CMP_RENF 139
CO 161
CO_LC 162
CO_LC_CP 164
CO_LC_ELI 163
CO_LC_VMAE 163
CO_LC_VMSE 163
CO_LC_WWM 165
CO_LC_WWS 164
CO_MC 162
CO_STD_ELI 161
COMT 107
COND 203
COND_DDL 204
COND_IMP 204
COND_NUL 205
COND_REP 206
COOR 110
CR1 238
CR2 238
CRG 238

D

DB 176
DB_CCH 176
DB_CP 178
DB_CPP 178
DB_EMP 177
DB_NLG 177
DEC_VA 214
DPL 239
DT 179
DT_CCH 180
DT_CP 181
DT_CPP 182
DT_EMP 180
DT_NLG 181
DTLI 222
DTNL 222
DTO 239
DYNI 224

E

EB 183

EB_CCH 184
EB_EMP 184
EB_NLG 184
ECR_CH 139
ECR_HSM 139
ECR_L 137
ECR_NMC 137
ECR_PH 137
ECR_PHM 137
ECR_V 138
ECR_VD 138
ECR_VM 137
ECR_VMV 138
ECR_VPP 138
ECR_VRS 139
EFD_COUC 210
EFD_INST 210
EFN 239
EJ 168
EJ_AD 168
EJ_EL 169
EJ_EPP 169
EJ_FC 168
EJ_GP 169
ELAS_ANL 132
ELAS_FC 131
ELAS_HSM 132
ELAS_LI 131
ELAS_LIV 131
ELEM 110
ENL 239
EST 240
ET 185
ET_CCH 185
ET_EMP 185
ET_NLG 186
EXEC 107
EXPO 108

F

FD 165
FD_AD 166
FD_FC 166
FD_GP 167
FLAM 224
FNC_C 133
FNC_CCM 134
FNC_CO 134
FNC_DP 134
FNC_DPC 135
FNC_HSM 135
FNC_MC 133
FNC_MCV 134
FNC_T 133
FNC_VM 133
FNC_VMC 135
FSC 240
FSR 241

G

GEFI 108
GEN_IIP 109
GEN_IMP 205
GEN_IRC 109
GEN_NUL 205
GEN_SOL 220
GFD 253

H

HPE 241

I

ILIG 107
IMPR 108
INA 242
INI 242
INP 244
INT 245
INU 245

J

JDD 104

K

KR 171
KR_ELI 172
KR_INT_EBP 173
KR_INT_EL 172
KR_INT_ELP 173
KR_INT_ERP 173
KVCOND 246

L

LAM_COUC 212
LAM_FD 212
LIM 246
LINC 225
LINE 225
LINH 226

M

MAIL 252

MB 112
MB_BAO 130
MB_BJA 115
MB_CCM 121
MB_CO 122
MB_CP 118
MB_DPAE 118
MB_DPSE 117
MB_EDI 129
MB_ELI 114
MB_ELO 114
MB_HB 122
MB_MCSE 115
MB_MCSE_EO 124
MB_ME 123
MB_NO 120
MB_PH 121
MB_RBS 126
MB_TA 125
MB_VE 119
MB_VMAE 117
MB_VMSE 116
MB_WWM 128
MB_WWS 127
MCNL 226
MEXO 228
MODE 229
MPLI 229
MPNL 230
MT 141
MT_BAO 153
MT_BJA 144
MT_CCM 148
MT_CO 149
MT_CP 146
MT_DPAE 146
MT_DPSE 145
MT_EDI 153
MT_ELI 143
MT_ELO 143
MT_HB 149
MT_MCSE 144
MT_NO 147
MT_PH 148
MT_RBS 150
MT_VE 147
MT_VMAE 145
MT_VMSE 145
MT_WWM 152
MT_WWS 152
MUL 247

N

NDP 247
NSAT 232

O

OB 187
OB_CCM 194
OB_CO 194
OB_CP 192
OB_DPAE 192
OB_DPSE 191
OB_ELI 188
OB_ELO 189
OB_MC 190
OB_NO 193
OB_PH 194
OB_VE 192
OB_VMAE 191
OB_VMSE 190
OT 195
OT_CCM 202
OT_CO 203
OT_CP 200
OT_DPAE 200
OT_DPSE 199
OT_ELI 196
OT_ELO 197
OT_NO 201
OT_PH 202
OT_VE 201
OT_VMAE 199
OT_VMSE 199

P

PB 154
PB_ELI 154
PNC_PAS 215
POI_FV 216
PT 155
PT_CF 160
PT_LF 157
PT_LF_CP 159
PT_LF_ELI 157
PT_LF_VMAE 158
PT_LF_VMSE 158
PT_LF_WWM 160
PT_LF_WWS 159

PT_MF 156
PT_STD_ELI 155
PTX 248

Q

QAB 248

R

RENF_DC 141
RENF_DS 141
RENF_EL 140
RENF_H 140
RENF_PP 140
RENF_R 140
RL 174
RSV4 254

S

SB 186
SIG_COUC 218
SIG_CST 217
SIG_GEO 218
SIG_ISO 219
SIG_THER 218
SP 174
SRE 249
STK 249
STP 250
STT 250
STU 250
SUI 251
SUMO 233
SURF 234

T

TCNL 234
TEST 106
TEXO 235
TXO 251

V

VPEG 240