

# OpenSeesPyAssistant

Generated by Doxygen 1.9.3

Author: Carmine Schipani

Date: January 21 2022



<b>1 OpenSeesPyAssistant</b>	<b>1</b>
<b>2 Namespace Index</b>	<b>3</b>
2.1 Packages	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Namespace Documentation</b>	<b>13</b>
6.1 AnalysisAndPostProcessing Namespace Reference	13
6.1.1 Detailed Description	13
6.2 Connections Namespace Reference	13
6.2.1 Detailed Description	14
6.2.2 Function Documentation	14
6.2.2.1 Pin()	14
6.2.2.2 RigidSupport()	14
6.2.2.3 RotationalSpring()	15
6.3 Constants Namespace Reference	16
6.3.1 Detailed Description	16
6.3.2 Variable Documentation	17
6.3.2.1 G_CONST	17
6.3.2.2 MAX_ITER	17
6.3.2.3 MAX_ITER_INTEGRATION	17
6.3.2.4 RIGID	17
6.3.2.5 TOL	17
6.3.2.6 TOL_INTEGRATION	18
6.3.2.7 ZERO	18
6.4 DataManagement Namespace Reference	18
6.4.1 Detailed Description	18
6.5 ErrorHandling Namespace Reference	18
6.5.1 Detailed Description	19
6.6 Fibers Namespace Reference	19
6.6.1 Detailed Description	20
6.6.2 Function Documentation	20
6.6.2.1 create_fiber_section()	20
6.6.2.2 plot_fiber_section()	22
6.7 FunctionalFeatures Namespace Reference	24
6.7.1 Detailed Description	25

6.7.2 Function Documentation	25
6.7.2.1 DiscretizeLinearly()	25
6.7.2.2 DiscretizeLoadProtocol()	27
6.7.2.3 GridIDConvention()	28
6.7.2.4 IDConvention()	30
6.7.2.5 NodesOrientation()	31
6.7.2.6 OffsetNodeIDConvention()	32
6.7.2.7 plot_member()	33
6.7.2.8 plot_nodes()	34
6.7.2.9 ProgressingPercentage()	35
6.8 GeometryTemplate Namespace Reference	36
6.8.1 Detailed Description	37
6.8.2 Function Documentation	37
6.8.2.1 DefineFrameNodes()	37
6.8.2.2 DefineFrameNodesAndElementsSteelShape()	39
6.8.2.3 DefineSubassemblageNodes()	41
6.8.2.4 Initialize2DModel()	43
6.9 MaterialModels Namespace Reference	43
6.9.1 Detailed Description	45
6.9.2 Function Documentation	45
6.9.2.1 Concrete01Funct()	46
6.9.2.2 Concrete04Funct()	47
6.9.2.3 PlotConcrete01()	48
6.9.2.4 PlotConcrete04()	49
6.10 MemberModel Namespace Reference	50
6.10.1 Detailed Description	51
6.10.2 Function Documentation	51
6.10.2.1 DefinePanelZoneElements()	52
6.10.2.2 DefinePanelZoneNodes()	53
6.11 Section Namespace Reference	54
6.11.1 Detailed Description	55
6.11.2 Function Documentation	55
6.11.2.1 ComputeACircle()	55
6.11.2.2 ComputeRho()	55
6.12 Units Namespace Reference	56
6.12.1 Detailed Description	57
6.12.2 Variable Documentation	57
6.12.2.1 cm2_unit	57
6.12.2.2 cm3_unit	58
6.12.2.3 cm4_unit	58
6.12.2.4 cm_unit	58
6.12.2.5 dm2_unit	58

6.12.2.6 dm3_unit . . . . .	58
6.12.2.7 dm4_unit . . . . .	58
6.12.2.8 dm_unit . . . . .	59
6.12.2.9 force_unit . . . . .	59
6.12.2.10 ft2_unit . . . . .	59
6.12.2.11 ft3_unit . . . . .	59
6.12.2.12 ft4_unit . . . . .	59
6.12.2.13 ft_unit . . . . .	59
6.12.2.14 GN_unit . . . . .	60
6.12.2.15 GPa_unit . . . . .	60
6.12.2.16 hours_unit . . . . .	60
6.12.2.17 inch2_unit . . . . .	60
6.12.2.18 inch3_unit . . . . .	60
6.12.2.19 inch4_unit . . . . .	60
6.12.2.20 inch_unit . . . . .	61
6.12.2.21 kg_unit . . . . .	61
6.12.2.22 kip_unit . . . . .	61
6.12.2.23 km_unit . . . . .	61
6.12.2.24 kN_unit . . . . .	61
6.12.2.25 kNm_unit . . . . .	61
6.12.2.26 kNmm_unit . . . . .	62
6.12.2.27 kPa_unit . . . . .	62
6.12.2.28 ksi_unit . . . . .	62
6.12.2.29 length_unit . . . . .	62
6.12.2.30 m2_unit . . . . .	62
6.12.2.31 m3_unit . . . . .	62
6.12.2.32 m4_unit . . . . .	63
6.12.2.33 m_unit . . . . .	63
6.12.2.34 mile_unit . . . . .	63
6.12.2.35 min_unit . . . . .	63
6.12.2.36 mm2_unit . . . . .	63
6.12.2.37 mm3_unit . . . . .	63
6.12.2.38 mm4_unit . . . . .	64
6.12.2.39 mm_unit . . . . .	64
6.12.2.40 MN_unit . . . . .	64
6.12.2.41 MNm_unit . . . . .	64
6.12.2.42 MNmm_unit . . . . .	64
6.12.2.43 MPa_unit . . . . .	64
6.12.2.44 N_unit . . . . .	65
6.12.2.45 Nm_unit . . . . .	65
6.12.2.46 Nmm_unit . . . . .	65
6.12.2.47 Pa_unit . . . . .	65

6.12.2.48 pound_unit . . . . .	65
6.12.2.49 psi_unit . . . . .	65
6.12.2.50 s_unit . . . . .	66
6.12.2.51 t_unit . . . . .	66
6.12.2.52 time_unit . . . . .	66
<b>7 Class Documentation</b>	<b>67</b>
7.1 Analysis Class Reference . . . . .	67
7.1.1 Detailed Description . . . . .	68
7.1.2 Constructor & Destructor Documentation . . . . .	68
7.1.2.1 __init__() . . . . .	68
7.1.3 Member Function Documentation . . . . .	70
7.1.3.1 DeformedShape() . . . . .	70
7.1.3.2 FiberResponse() . . . . .	71
7.1.3.3 Gravity() . . . . .	72
7.1.3.4 LateralForce() . . . . .	74
7.1.3.5 LoadingProtocol() . . . . .	77
7.1.3.6 Pushover() . . . . .	79
7.1.4 Member Data Documentation . . . . .	82
7.1.4.1 algo . . . . .	82
7.1.4.2 allow_smaller_step . . . . .	82
7.1.4.3 data_dir . . . . .	82
7.1.4.4 load_case . . . . .	83
7.1.4.5 max_iter . . . . .	83
7.1.4.6 name_ODB . . . . .	83
7.1.4.7 test_opt . . . . .	83
7.1.4.8 test_type . . . . .	83
7.1.4.9 tol . . . . .	83
7.2 ConfMander1988Circ Class Reference . . . . .	84
7.2.1 Detailed Description . . . . .	85
7.2.2 Constructor & Destructor Documentation . . . . .	85
7.2.2.1 __init__() . . . . .	86
7.2.3 Member Function Documentation . . . . .	88
7.2.3.1 CheckApplicability() . . . . .	88
7.2.3.2 Compute_ec() . . . . .	89
7.2.3.3 Compute_ecc() . . . . .	89
7.2.3.4 Compute_eccu() . . . . .	90
7.2.3.5 Compute_ecp() . . . . .	90
7.2.3.6 Compute_ecu() . . . . .	91
7.2.3.7 Compute_et() . . . . .	91
7.2.3.8 Compute_fct() . . . . .	92
7.2.3.9 Concrete01() . . . . .	92

7.2.3.10 Concrete04()	93
7.2.3.11 ReInit()	93
7.2.3.12 ShowInfo()	94
7.2.3.13 UpdateStoredData()	95
7.2.4 Member Data Documentation	95
7.2.4.1 Ac	95
7.2.4.2 Acc	96
7.2.4.3 Ae	96
7.2.4.4 bc	96
7.2.4.5 beta	96
7.2.4.6 D_bars	96
7.2.4.7 D_hoops	96
7.2.4.8 data	97
7.2.4.9 Ec	97
7.2.4.10 ec	97
7.2.4.11 ecc	97
7.2.4.12 eccu	97
7.2.4.13 ecp	97
7.2.4.14 ecu	98
7.2.4.15 esu	98
7.2.4.16 et	98
7.2.4.17 fc	98
7.2.4.18 fcc	98
7.2.4.19 fct	98
7.2.4.20 fl	99
7.2.4.21 fl_prime	99
7.2.4.22 fs	99
7.2.4.23 ID	99
7.2.4.24 Initialized	99
7.2.4.25 K_combo	99
7.2.4.26 ke	100
7.2.4.27 nr_bars	100
7.2.4.28 rho_cc	100
7.2.4.29 rho_s_vol	100
7.2.4.30 s	100
7.2.4.31 section_name_tag	100
7.3 ConfMander1988CircRCCircShape Class Reference	101
7.3.1 Detailed Description	101
7.3.2 Constructor & Destructor Documentation	101
7.3.2.1 __init__()	102
7.3.3 Member Data Documentation	103
7.3.3.1 section	103

---

7.3.3.2 section_name_tag . . . . .	103
7.4 ConfMander1988Rect Class Reference . . . . .	103
7.4.1 Detailed Description . . . . .	105
7.4.2 Constructor & Destructor Documentation . . . . .	105
7.4.2.1 __init__() . . . . .	106
7.4.3 Member Function Documentation . . . . .	109
7.4.3.1 CheckApplicability() . . . . .	109
7.4.3.2 Compute_ec() . . . . .	109
7.4.3.3 Compute_ecc() . . . . .	110
7.4.3.4 Compute_eccu() . . . . .	110
7.4.3.5 Compute_ecp() . . . . .	111
7.4.3.6 Compute_ecu() . . . . .	111
7.4.3.7 Compute_et() . . . . .	112
7.4.3.8 Compute_fct() . . . . .	112
7.4.3.9 ComputeAi() . . . . .	113
7.4.3.10 ComputeConfinementFactor() . . . . .	113
7.4.3.11 Concrete01() . . . . .	114
7.4.3.12 Concrete04() . . . . .	115
7.4.3.13 ReInit() . . . . .	115
7.4.3.14 ShowInfo() . . . . .	116
7.4.3.15 UpdateStoredData() . . . . .	117
7.4.4 Member Data Documentation . . . . .	118
7.4.4.1 Ac . . . . .	118
7.4.4.2 Acc . . . . .	118
7.4.4.3 Ae . . . . .	118
7.4.4.4 Ai . . . . .	118
7.4.4.5 array_fl2 . . . . .	119
7.4.4.6 bc . . . . .	119
7.4.4.7 beta . . . . .	119
7.4.4.8 curve_fl1 . . . . .	119
7.4.4.9 DBars . . . . .	119
7.4.4.10 D_hoops . . . . .	119
7.4.4.11 data . . . . .	120
7.4.4.12 dc . . . . .	120
7.4.4.13 Ec . . . . .	120
7.4.4.14 ec . . . . .	120
7.4.4.15 ecc . . . . .	120
7.4.4.16 eccu . . . . .	120
7.4.4.17 ecp . . . . .	121
7.4.4.18 ecu . . . . .	121
7.4.4.19 esu . . . . .	121
7.4.4.20 et . . . . .	121



7.4.4.21 fc	121
7.4.4.22 fcc	121
7.4.4.23 fct	122
7.4.4.24 fl_x	122
7.4.4.25 fl_y	122
7.4.4.26 fs	122
7.4.4.27 ID	122
7.4.4.28 Initialized	122
7.4.4.29 K_combo	123
7.4.4.30 ke	123
7.4.4.31 nrBars	123
7.4.4.32 rho_cc	123
7.4.4.33 rho_s_x	123
7.4.4.34 rho_s_y	123
7.4.4.35 s	124
7.4.4.36 section_name_tag	124
7.4.4.37 wx_bottom	124
7.4.4.38 wx_top	124
7.4.4.39 wy	124
7.5 ConfMander1988RectRCRectShape Class Reference	125
7.5.1 Detailed Description	125
7.5.2 Constructor & Destructor Documentation	125
7.5.2.1 __init__()	126
7.5.3 Member Data Documentation	127
7.5.3.1 section	127
7.5.3.2 section_name_tag	127
7.6 DataManagement Class Reference	127
7.6.1 Detailed Description	128
7.6.2 Member Function Documentation	128
7.6.2.1 Relnit()	128
7.6.2.2 SaveData()	129
7.6.2.3 ShowInfo()	129
7.6.2.4 UpdateStoredData()	130
7.7 ElasticElement Class Reference	130
7.7.1 Detailed Description	131
7.7.2 Constructor & Destructor Documentation	131
7.7.2.1 __init__()	132
7.7.3 Member Function Documentation	133
7.7.3.1 CreateMember()	133
7.7.3.2 Record()	133
7.7.3.3 RecordNodeDef()	134
7.7.3.4 Relnit()	134

7.7.3.5 ShowInfo()	135
7.7.3.6 UpdateStoredData()	136
7.7.4 Member Data Documentation	136
7.7.4.1 A	136
7.7.4.2 data	136
7.7.4.3 E	136
7.7.4.4 element_array	137
7.7.4.5 element_ID	137
7.7.4.6 geo_transf_ID	137
7.7.4.7 Initialized	137
7.7.4.8 iNode_ID	137
7.7.4.9 ly	137
7.7.4.10 jNode_ID	138
7.7.4.11 section_name_tag	138
7.8 ElasticElementSteelShape Class Reference	138
7.8.1 Detailed Description	138
7.8.2 Constructor & Destructor Documentation	139
7.8.2.1 __init__()	139
7.8.3 Member Data Documentation	139
7.8.3.1 section	140
7.8.3.2 section_name_tag	140
7.9 Fibers Class Reference	140
7.9.1 Detailed Description	140
7.10 FibersCirc Class Reference	141
7.10.1 Detailed Description	142
7.10.2 Constructor & Destructor Documentation	142
7.10.2.1 __init__()	142
7.10.3 Member Function Documentation	144
7.10.3.1 CreateFibers()	144
7.10.3.2 Relnit()	145
7.10.3.3 ShowInfo()	145
7.10.3.4 UpdateStoredData()	146
7.10.4 Member Data Documentation	146
7.10.4.1 alpha_i	147
7.10.4.2 Ay	147
7.10.4.3 b	147
7.10.4.4 bars_mat_ID	147
7.10.4.5 conf_mat_ID	147
7.10.4.6 DBars	147
7.10.4.7 D_hoops	148
7.10.4.8 data	148
7.10.4.9 discr_core	148

7.10.4.10	<a href="#">discr_cover</a>	148
7.10.4.11	<a href="#">e</a>	148
7.10.4.12	<a href="#">fib_sec</a>	148
7.10.4.13	<a href="#">GJ</a>	149
7.10.4.14	<a href="#">ID</a>	149
7.10.4.15	<a href="#">Initialized</a>	149
7.10.4.16	<a href="#">nBars</a>	149
7.10.4.17	<a href="#">rBars</a>	149
7.10.4.18	<a href="#">r_core</a>	149
7.10.4.19	<a href="#">section_name_tag</a>	150
7.10.4.20	<a href="#">unconf_mat_ID</a>	150
7.11	<a href="#">FibersCircRCCircShape Class Reference</a>	150
7.11.1	<a href="#">Detailed Description</a>	150
7.11.2	<a href="#">Constructor &amp; Destructor Documentation</a>	151
7.11.2.1	<a href="#">__init__()</a>	151
7.11.3	<a href="#">Member Data Documentation</a>	152
7.11.3.1	<a href="#">section</a>	152
7.11.3.2	<a href="#">section_name_tag</a>	152
7.12	<a href="#">FibersIShape Class Reference</a>	152
7.12.1	<a href="#">Detailed Description</a>	153
7.12.2	<a href="#">Constructor &amp; Destructor Documentation</a>	154
7.12.2.1	<a href="#">__init__()</a>	154
7.12.3	<a href="#">Member Function Documentation</a>	156
7.12.3.1	<a href="#">CreateFibers()</a>	156
7.12.3.2	<a href="#">Relnit()</a>	156
7.12.3.3	<a href="#">ShowInfo()</a>	157
7.12.3.4	<a href="#">UpdateStoredData()</a>	158
7.12.4	<a href="#">Member Data Documentation</a>	158
7.12.4.1	<a href="#">bf_b</a>	158
7.12.4.2	<a href="#">bf_t</a>	158
7.12.4.3	<a href="#">bottom_flange_mat_ID</a>	159
7.12.4.4	<a href="#">d</a>	159
7.12.4.5	<a href="#">data</a>	159
7.12.4.6	<a href="#">discr_bottom_flange</a>	159
7.12.4.7	<a href="#">discr_top_flange</a>	159
7.12.4.8	<a href="#">discr_web</a>	159
7.12.4.9	<a href="#">fib_sec</a>	160
7.12.4.10	<a href="#">GJ</a>	160
7.12.4.11	<a href="#">ID</a>	160
7.12.4.12	<a href="#">Initialized</a>	160
7.12.4.13	<a href="#">section_name_tag</a>	160
7.12.4.14	<a href="#">tf_b</a>	160

7.12.4.15 tf_t	161
7.12.4.16 top_flange_mat_ID	161
7.12.4.17 tw	161
7.12.4.18 web_mat_ID	161
7.13 FibersIShapeSteelIShape Class Reference	161
7.13.1 Detailed Description	162
7.13.2 Constructor & Destructor Documentation	162
7.13.2.1 __init__()	162
7.13.3 Member Data Documentation	163
7.13.3.1 section	163
7.13.3.2 section_name_tag	164
7.14 FibersRect Class Reference	164
7.14.1 Detailed Description	165
7.14.2 Constructor & Destructor Documentation	165
7.14.2.1 __init__()	165
7.14.3 Member Function Documentation	168
7.14.3.1 CreateFibers()	168
7.14.3.2 Relnit()	169
7.14.3.3 ShowInfo()	169
7.14.3.4 UpdateStoredData()	170
7.14.4 Member Data Documentation	171
7.14.4.1 Ay	171
7.14.4.2 b	171
7.14.4.3 bars_mat_ID	171
7.14.4.4 bars_x	171
7.14.4.5 conf_mat_ID	171
7.14.4.6 d	172
7.14.4.7 D_hoops	172
7.14.4.8 data	172
7.14.4.9 discr_core	172
7.14.4.10 discr_cover_lateral	172
7.14.4.11 discr_cover_topbottom	172
7.14.4.12 e	173
7.14.4.13 fib_sec	173
7.14.4.14 GJ	173
7.14.4.15 ID	173
7.14.4.16 Initialized	173
7.14.4.17 ranges_y	173
7.14.4.18 rebarYZ	174
7.14.4.19 section_name_tag	174
7.14.4.20 unconf_mat_ID	174
7.15 FibersRectRCRectShape Class Reference	174

7.15.1 Detailed Description . . . . .	175
7.15.2 Constructor & Destructor Documentation . . . . .	175
7.15.2.1 __init__() . . . . .	175
7.15.3 Member Data Documentation . . . . .	176
7.15.3.1 section . . . . .	176
7.15.3.2 section_name_tag . . . . .	176
7.16 ForceBasedElement Class Reference . . . . .	177
7.16.1 Detailed Description . . . . .	177
7.16.2 Constructor & Destructor Documentation . . . . .	178
7.16.2.1 __init__() . . . . .	178
7.16.3 Member Function Documentation . . . . .	180
7.16.3.1 CreateMember() . . . . .	180
7.16.3.2 Record() . . . . .	180
7.16.3.3 RecordNodeDef() . . . . .	181
7.16.3.4 ReInit() . . . . .	181
7.16.3.5 ShowInfo() . . . . .	182
7.16.3.6 UpdateStoredData() . . . . .	182
7.16.4 Member Data Documentation . . . . .	183
7.16.4.1 data . . . . .	183
7.16.4.2 element_array . . . . .	183
7.16.4.3 element_ID . . . . .	183
7.16.4.4 fiber_ID . . . . .	183
7.16.4.5 geo_transf_ID . . . . .	184
7.16.4.6 Initialized . . . . .	184
7.16.4.7 iNode_ID . . . . .	184
7.16.4.8 integration_type . . . . .	184
7.16.4.9 lp . . . . .	184
7.16.4.10 jNode_ID . . . . .	184
7.16.4.11 max_iter . . . . .	185
7.16.4.12 new_integration_ID . . . . .	185
7.16.4.13 section_name_tag . . . . .	185
7.16.4.14 tol . . . . .	185
7.17 ForceBasedElementFibersCircRCCircShape Class Reference . . . . .	185
7.17.1 Detailed Description . . . . .	186
7.17.2 Constructor & Destructor Documentation . . . . .	186
7.17.2.1 __init__() . . . . .	186
7.17.3 Member Data Documentation . . . . .	187
7.17.3.1 section . . . . .	187
7.17.3.2 section_name_tag . . . . .	188
7.18 ForceBasedElementFibersIShapeSteelIShape Class Reference . . . . .	188
7.18.1 Detailed Description . . . . .	188
7.18.2 Constructor & Destructor Documentation . . . . .	189

7.18.2.1 <code>__init__()</code>	189
7.18.3 Member Data Documentation	190
7.18.3.1 section	190
7.18.3.2 section_name_tag	190
7.19 ForceBasedElementFibersRectRCRectShape Class Reference	191
7.19.1 Detailed Description	191
7.19.2 Constructor & Destructor Documentation	191
7.19.2.1 <code>__init__()</code>	192
7.19.3 Member Data Documentation	193
7.19.3.1 section	193
7.19.3.2 section_name_tag	193
7.20 GIFBElement Class Reference	193
7.20.1 Detailed Description	194
7.20.2 Constructor & Destructor Documentation	195
7.20.2.1 <code>__init__()</code>	195
7.20.3 Member Function Documentation	197
7.20.3.1 <code>ComputeIp()</code>	197
7.20.3.2 <code>ComputeLp()</code>	198
7.20.3.3 <code>CreateMember()</code>	198
7.20.3.4 <code>Record()</code>	199
7.20.3.5 <code>RecordNodeDef()</code>	199
7.20.3.6 <code>Relnit()</code>	199
7.20.3.7 <code>ShowInfo()</code>	200
7.20.3.8 <code>UpdateStoredData()</code>	201
7.20.4 Member Data Documentation	202
7.20.4.1 <code>D_bars</code>	202
7.20.4.2 <code>data</code>	202
7.20.4.3 <code>element_array</code>	202
7.20.4.4 <code>element_ID</code>	202
7.20.4.5 <code>fiber_ID</code>	202
7.20.4.6 <code>fy</code>	203
7.20.4.7 <code>geo_transf_ID</code>	203
7.20.4.8 <code>Initialized</code>	203
7.20.4.9 <code>iNode_ID</code>	203
7.20.4.10 <code>Ip</code>	203
7.20.4.11 <code>jNode_ID</code>	203
7.20.4.12 <code>L</code>	204
7.20.4.13 <code>lambda_i</code>	204
7.20.4.14 <code>lambda_j</code>	204
7.20.4.15 <code>Lp</code>	204
7.20.4.16 <code>max_iter</code>	204
7.20.4.17 <code>max_tol</code>	204

7.20.4.18 min_tol . . . . .	205
7.20.4.19 new_integration_ID . . . . .	205
7.20.4.20 section_name_tag . . . . .	205
7.21 GIFBELEMENTFibersCircRCCircShape Class Reference . . . . .	205
7.21.1 Detailed Description . . . . .	206
7.21.2 Constructor & Destructor Documentation . . . . .	206
7.21.2.1 __init__() . . . . .	206
7.21.3 Member Data Documentation . . . . .	207
7.21.3.1 section . . . . .	208
7.21.3.2 section_name_tag . . . . .	208
7.22 GIFBELEMENTFibersRectRCRectShape Class Reference . . . . .	208
7.22.1 Detailed Description . . . . .	208
7.22.2 Constructor & Destructor Documentation . . . . .	209
7.22.2.1 __init__() . . . . .	209
7.22.3 Member Data Documentation . . . . .	210
7.22.3.1 section . . . . .	210
7.22.3.2 section_name_tag . . . . .	210
7.23 GIFBELEMENTRCCircShape Class Reference . . . . .	211
7.23.1 Detailed Description . . . . .	211
7.23.2 Constructor & Destructor Documentation . . . . .	211
7.23.2.1 __init__() . . . . .	212
7.23.3 Member Data Documentation . . . . .	213
7.23.3.1 section . . . . .	213
7.23.3.2 section_name_tag . . . . .	213
7.24 GIFBELEMENTRCRectShape Class Reference . . . . .	214
7.24.1 Detailed Description . . . . .	214
7.24.2 Constructor & Destructor Documentation . . . . .	214
7.24.2.1 __init__() . . . . .	215
7.24.3 Member Data Documentation . . . . .	216
7.24.3.1 section . . . . .	216
7.24.3.2 section_name_tag . . . . .	216
7.25 GMP1970 Class Reference . . . . .	217
7.25.1 Detailed Description . . . . .	218
7.25.2 Constructor & Destructor Documentation . . . . .	219
7.25.2.1 __init__() . . . . .	219
7.25.3 Member Function Documentation . . . . .	220
7.25.3.1 CheckApplicability() . . . . .	220
7.25.3.2 Relnit() . . . . .	221
7.25.3.3 ShowInfo() . . . . .	221
7.25.3.4 Steel02() . . . . .	222
7.25.3.5 UpdateStoredData() . . . . .	222
7.25.4 Member Data Documentation . . . . .	223

7.25.4.1 a1	223
7.25.4.2 a2	223
7.25.4.3 a3	223
7.25.4.4 a4	223
7.25.4.5 b	223
7.25.4.6 cR1	224
7.25.4.7 cR2	224
7.25.4.8 data	224
7.25.4.9 Ey	224
7.25.4.10 fy	224
7.25.4.11 ID	224
7.25.4.12 Initialized	225
7.25.4.13 R0	225
7.25.4.14 section_name_tag	225
7.26 GMP1970RCRectShape Class Reference	225
7.26.1 Detailed Description	226
7.26.2 Constructor & Destructor Documentation	226
7.26.2.1 __init__()	226
7.26.3 Member Data Documentation	227
7.26.3.1 section	227
7.26.3.2 section_name_tag	228
7.27 Gupta1999 Class Reference	228
7.27.1 Detailed Description	229
7.27.2 Constructor & Destructor Documentation	230
7.27.2.1 __init__()	230
7.27.3 Member Function Documentation	232
7.27.3.1 CheckApplicability()	232
7.27.3.2 Hysteretic()	232
7.27.3.3 ReInit()	233
7.27.3.4 ShowInfo()	233
7.27.3.5 UpdateStoredData()	234
7.27.4 Member Data Documentation	235
7.27.4.1 a_s	235
7.27.4.2 beam_section_name_tag	235
7.27.4.3 beta	235
7.27.4.4 bf_c	236
7.27.4.5 col_section_name_tag	236
7.27.4.6 d_b	236
7.27.4.7 d_c	236
7.27.4.8 data	236
7.27.4.9 dmg1	236
7.27.4.10 dmg2	237



7.27.4.11 E	237
7.27.4.12 Fy	237
7.27.4.13 G	237
7.27.4.14 gamma1_y	237
7.27.4.15 gamma2_y	237
7.27.4.16 gamma3_y	238
7.27.4.17 I_c	238
7.27.4.18 ID	238
7.27.4.19 Initialized	238
7.27.4.20 Ke	238
7.27.4.21 Kp	238
7.27.4.22 M1y	239
7.27.4.23 M2y	239
7.27.4.24 M3y	239
7.27.4.25 pinchx	239
7.27.4.26 pinchy	239
7.27.4.27 Ry	239
7.27.4.28 t_dp	240
7.27.4.29 t_p	240
7.27.4.30 t_pz	240
7.27.4.31 tf_b	240
7.27.4.32 tf_c	240
7.27.4.33 Vy	240
7.28 Gupta1999SteelShape Class Reference	241
7.28.1 Detailed Description	241
7.28.2 Constructor & Destructor Documentation	241
7.28.2.1 __init__()	242
7.28.3 Member Data Documentation	243
7.28.3.1 beam	243
7.28.3.2 beam_section_name_tag	243
7.28.3.3 col	243
7.28.3.4 col_section_name_tag	243
7.29 IDGenerator Class Reference	244
7.29.1 Detailed Description	244
7.29.2 Constructor & Destructor Documentation	244
7.29.2.1 __init__()	244
7.29.3 Member Function Documentation	245
7.29.3.1 GenerateIDElement()	245
7.29.3.2 GenerateIDFiber()	245
7.29.3.3 GenerateIDMat()	246
7.29.3.4 GenerateIDNode()	246
7.29.4 Member Data Documentation	246

7.29.4.1 current_element_ID . . . . .	246
7.29.4.2 current_fiber_ID . . . . .	247
7.29.4.3 current_mat_ID . . . . .	247
7.29.4.4 current_node_ID . . . . .	247
7.30 InconsistentGeometry Class Reference . . . . .	247
7.30.1 Detailed Description . . . . .	247
7.31 MaterialModels Class Reference . . . . .	248
7.31.1 Detailed Description . . . . .	248
7.31.2 Member Function Documentation . . . . .	249
7.31.2.1 CheckApplicability() . . . . .	249
7.32 MemberFailure Class Reference . . . . .	249
7.32.1 Detailed Description . . . . .	249
7.33 MemberModel Class Reference . . . . .	250
7.33.1 Detailed Description . . . . .	250
7.33.2 Member Function Documentation . . . . .	250
7.33.2.1 Record() . . . . .	250
7.33.2.2 RecordNodeDef() . . . . .	251
7.34 ModifiedIMK Class Reference . . . . .	252
7.34.1 Detailed Description . . . . .	254
7.34.2 Constructor & Destructor Documentation . . . . .	254
7.34.2.1 __init__() . . . . .	255
7.34.3 Member Function Documentation . . . . .	258
7.34.3.1 Bilin() . . . . .	258
7.34.3.2 CheckApplicability() . . . . .	258
7.34.3.3 Computea() . . . . .	259
7.34.3.4 Computea_s() . . . . .	259
7.34.3.5 ComputeK() . . . . .	260
7.34.3.6 ComputeKe() . . . . .	260
7.34.3.7 ComputeMc() . . . . .	261
7.34.3.8 ComputeMyStar() . . . . .	261
7.34.3.9 ComputeRefEnergyDissipationCap() . . . . .	262
7.34.3.10 ComputeTheta_p() . . . . .	262
7.34.3.11 ComputeTheta_pc() . . . . .	263
7.34.3.12 ComputeTheta_u() . . . . .	264
7.34.3.13 ComputeTheta_y() . . . . .	264
7.34.3.14 ReInit() . . . . .	264
7.34.3.15 ShowInfo() . . . . .	265
7.34.3.16 UpdateStoredData() . . . . .	266
7.34.4 Member Data Documentation . . . . .	267
7.34.4.1 a . . . . .	267
7.34.4.2 a_s . . . . .	267
7.34.4.3 bf . . . . .	267

7.34.4.4 d	268
7.34.4.5 data	268
7.34.4.6 E	268
7.34.4.7 Fy	268
7.34.4.8 gamma_rm	268
7.34.4.9 h_1	268
7.34.4.10 ID	269
7.34.4.11 Initialized	269
7.34.4.12 ly_mod	269
7.34.4.13 iz	269
7.34.4.14 K	269
7.34.4.15 K_factor	269
7.34.4.16 Ke	270
7.34.4.17 L	270
7.34.4.18 L_0	270
7.34.4.19 L_b	270
7.34.4.20 Mc	270
7.34.4.21 McMy	270
7.34.4.22 My	271
7.34.4.23 My_star	271
7.34.4.24 n	271
7.34.4.25 N_G	271
7.34.4.26 Npl	271
7.34.4.27 prob_factor	271
7.34.4.28 rate_det	272
7.34.4.29 section_name_tag	272
7.34.4.30 tf	272
7.34.4.31 theta_p	272
7.34.4.32 theta_pc	272
7.34.4.33 theta_u	272
7.34.4.34 theta_y	273
7.34.4.35 tw	273
7.34.4.36 Type	273
7.35 ModifiedIMKSteelShape Class Reference	273
7.35.1 Detailed Description	274
7.35.2 Constructor & Destructor Documentation	274
7.35.2.1 __init__()	274
7.35.3 Member Data Documentation	275
7.35.3.1 section	275
7.35.3.2 section_name_tag	276
7.36 NegativeValue Class Reference	276
7.36.1 Detailed Description	276

7.37 NoApplicability Class Reference . . . . .	276
7.37.1 Detailed Description . . . . .	277
7.38 PanelZone Class Reference . . . . .	277
7.38.1 Detailed Description . . . . .	278
7.38.2 Constructor & Destructor Documentation . . . . .	278
7.38.2.1 __init__() . . . . .	278
7.38.3 Member Function Documentation . . . . .	280
7.38.3.1 CreateMember() . . . . .	280
7.38.3.2 Record() . . . . .	281
7.38.3.3 RecordNodeDef() . . . . .	281
7.38.3.4 ReInit() . . . . .	282
7.38.3.5 ShowInfo() . . . . .	282
7.38.3.6 UpdateStoredData() . . . . .	283
7.38.4 Member Data Documentation . . . . .	283
7.38.4.1 A_rigid . . . . .	283
7.38.4.2 beam_section_name_tag . . . . .	284
7.38.4.3 col_section_name_tag . . . . .	284
7.38.4.4 data . . . . .	284
7.38.4.5 E . . . . .	284
7.38.4.6 element_array . . . . .	284
7.38.4.7 geo_transf_ID . . . . .	284
7.38.4.8 I_rigid . . . . .	285
7.38.4.9 Initialized . . . . .	285
7.38.4.10 iNode_ID . . . . .	285
7.38.4.11 jNode_ID . . . . .	285
7.38.4.12 master_node_ID . . . . .	285
7.38.4.13 mat_ID . . . . .	285
7.38.4.14 mid_panel_zone_height . . . . .	286
7.38.4.15 mid_panel_zone_width . . . . .	286
7.38.4.16 pin_corners . . . . .	286
7.38.4.17 spring_ID . . . . .	286
7.39 PanelZoneRCS Class Reference . . . . .	286
7.39.1 Detailed Description . . . . .	287
7.39.2 Constructor & Destructor Documentation . . . . .	287
7.39.2.1 __init__() . . . . .	287
7.39.3 Member Data Documentation . . . . .	288
7.39.3.1 beam . . . . .	288
7.39.3.2 beam_section_name_tag . . . . .	288
7.39.3.3 col . . . . .	288
7.39.3.4 col_section_name_tag . . . . .	289
7.40 PanelZoneSteelShape Class Reference . . . . .	289
7.40.1 Detailed Description . . . . .	289

7.40.2 Constructor & Destructor Documentation	290
7.40.2.1 <code>__init__()</code>	290
7.40.3 Member Data Documentation	291
7.40.3.1 <code>beam</code>	291
7.40.3.2 <code>beam_section_name_tag</code>	291
7.40.3.3 <code>col</code>	291
7.40.3.4 <code>col_section_name_tag</code>	291
7.41 PanelZoneSteelShapeGupta1999 Class Reference	292
7.41.1 Detailed Description	292
7.41.2 Constructor & Destructor Documentation	292
7.41.2.1 <code>__init__()</code>	293
7.41.3 Member Data Documentation	293
7.41.3.1 <code>beam</code>	293
7.41.3.2 <code>col</code>	294
7.42 PanelZoneSteelShapeSkiadopoulos2021 Class Reference	294
7.42.1 Detailed Description	294
7.42.2 Constructor & Destructor Documentation	295
7.42.2.1 <code>__init__()</code>	295
7.42.3 Member Data Documentation	296
7.42.3.1 <code>beam</code>	296
7.42.3.2 <code>col</code>	296
7.43 PositiveValue Class Reference	296
7.43.1 Detailed Description	296
7.44 RCCCircShape Class Reference	297
7.44.1 Detailed Description	298
7.44.2 Constructor & Destructor Documentation	298
7.44.2.1 <code>__init__()</code>	298
7.44.3 Member Function Documentation	300
7.44.3.1 <code>ComputeEc()</code>	300
7.44.3.2 <code>ComputeI()</code>	301
7.44.3.3 <code>ComputeRhoVol()</code>	301
7.44.3.4 <code>ReInit()</code>	302
7.44.3.5 <code>ShowInfo()</code>	302
7.44.3.6 <code>UpdateStoredData()</code>	303
7.44.4 Member Data Documentation	303
7.44.4.1 <code>A</code>	304
7.44.4.2 <code>Ac</code>	304
7.44.4.3 <code>As</code>	304
7.44.4.4 <code>Ay</code>	304
7.44.4.5 <code>b</code>	304
7.44.4.6 <code>bc</code>	304
7.44.4.7 <code>cl_bars</code>	305

7.44.4.8 cl_hoops	305
7.44.4.9 D_bars	305
7.44.4.10 D_hoops	305
7.44.4.11 data	305
7.44.4.12 e	305
7.44.4.13 Ec	306
7.44.4.14 Es	306
7.44.4.15 Ey	306
7.44.4.16 fc	306
7.44.4.17 fs	306
7.44.4.18 fy	306
7.44.4.19 l	307
7.44.4.20 L	307
7.44.4.21 n_bars	307
7.44.4.22 name_tag	307
7.44.4.23 rho_bars	307
7.44.4.24 rho_s_vol	307
7.44.4.25 s	308
7.45 RRectShape Class Reference	308
7.45.1 Detailed Description	309
7.45.2 Constructor & Destructor Documentation	309
7.45.2.1 __init__()	310
7.45.3 Member Function Documentation	313
7.45.3.1 ComputeA()	313
7.45.3.2 ComputeAc()	313
7.45.3.3 ComputeEc()	314
7.45.3.4 Computely()	314
7.45.3.5 Computelz()	314
7.45.3.6 ComputeNrBars()	315
7.45.3.7 Relnit()	315
7.45.3.8 ShowInfo()	316
7.45.3.9 UpdateStoredData()	317
7.45.4 Member Data Documentation	317
7.45.4.1 A	317
7.45.4.2 Ac	317
7.45.4.3 As	318
7.45.4.4 Ay	318
7.45.4.5 b	318
7.45.4.6 bars_position_x	318
7.45.4.7 bars_ranges_position_y	318
7.45.4.8 bc	318
7.45.4.9 cl_bars	319

7.45.4.10 cl_hoops	319
7.45.4.11 d	319
7.45.4.12 D_bars	319
7.45.4.13 D_hoops	319
7.45.4.14 data	319
7.45.4.15 dc	320
7.45.4.16 e	320
7.45.4.17 Ec	320
7.45.4.18 Es	320
7.45.4.19 Ey	320
7.45.4.20 fc	320
7.45.4.21 fs	321
7.45.4.22 fy	321
7.45.4.23 ly	321
7.45.4.24 lz	321
7.45.4.25 L	321
7.45.4.26 name_tag	321
7.45.4.27 nr_bars	322
7.45.4.28 rho_bars	322
7.45.4.29 rho_s_x	322
7.45.4.30 rho_s_y	322
7.45.4.31 s	322
7.46 RCSquareShape Class Reference	323
7.46.1 Detailed Description	323
7.46.2 Constructor & Destructor Documentation	323
7.46.2.1 __init__()	323
7.47 Section Class Reference	325
7.47.1 Detailed Description	325
7.48 Skiadopoulos2021 Class Reference	326
7.48.1 Detailed Description	328
7.48.2 Constructor & Destructor Documentation	328
7.48.2.1 __init__()	328
7.48.3 Member Function Documentation	331
7.48.3.1 CheckApplicability()	331
7.48.3.2 Hysteretic()	332
7.48.3.3 ReInit()	332
7.48.3.4 ShowInfo()	333
7.48.3.5 UpdateStoredData()	334
7.48.4 Member Data Documentation	335
7.48.4.1 a_s	335
7.48.4.2 beam_section_name_tag	335
7.48.4.3 beta	335

7.48.4.4 bf_c	335
7.48.4.5 Cf1	336
7.48.4.6 Cf1_tests	336
7.48.4.7 Cf4	336
7.48.4.8 Cf4_tests	336
7.48.4.9 Cf6	336
7.48.4.10 Cf6_tests	336
7.48.4.11 col_section_name_tag	337
7.48.4.12 Cw1	337
7.48.4.13 Cw1_tests	337
7.48.4.14 Cw4	337
7.48.4.15 Cw4_tests	337
7.48.4.16 Cw6	337
7.48.4.17 Cw6_tests	338
7.48.4.18 d_b	338
7.48.4.19 d_c	338
7.48.4.20 data	338
7.48.4.21 dmg1	338
7.48.4.22 dmg2	338
7.48.4.23 E	339
7.48.4.24 Fy	339
7.48.4.25 G	339
7.48.4.26 Gamma_1	339
7.48.4.27 Gamma_4	339
7.48.4.28 Gamma_6	339
7.48.4.29 I_c	340
7.48.4.30 ID	340
7.48.4.31 Initialized	340
7.48.4.32 Kb	340
7.48.4.33 Kbf	340
7.48.4.34 Ke	340
7.48.4.35 Kf	341
7.48.4.36 Kf_Ke	341
7.48.4.37 Kf_Ke_tests	341
7.48.4.38 Ks	341
7.48.4.39 Ksf	341
7.48.4.40 M1	341
7.48.4.41 M4	342
7.48.4.42 M6	342
7.48.4.43 pinchx	342
7.48.4.44 pinchy	342
7.48.4.45 Ry	342



7.48.4.46 t_dp . . . . .	342
7.48.4.47 t_fbp . . . . .	343
7.48.4.48 t_p . . . . .	343
7.48.4.49 t_pz . . . . .	343
7.48.4.50 tf_b . . . . .	343
7.48.4.51 tf_c . . . . .	343
7.48.4.52 V1 . . . . .	343
7.48.4.53 V4 . . . . .	344
7.48.4.54 V6 . . . . .	344
7.49 Skiadopoulos2021RCS Class Reference . . . . .	344
7.49.1 Detailed Description . . . . .	344
7.49.2 Constructor & Destructor Documentation . . . . .	345
7.49.2.1 __init__() . . . . .	345
7.49.3 Member Data Documentation . . . . .	346
7.49.3.1 beam . . . . .	346
7.49.3.2 beam_section_name_tag . . . . .	346
7.50 Skiadopoulos2021SteelShape Class Reference . . . . .	347
7.50.1 Detailed Description . . . . .	347
7.50.2 Constructor & Destructor Documentation . . . . .	348
7.50.2.1 __init__() . . . . .	348
7.50.3 Member Data Documentation . . . . .	349
7.50.3.1 beam . . . . .	349
7.50.3.2 beam_section_name_tag . . . . .	349
7.50.3.3 col . . . . .	349
7.50.3.4 col_section_name_tag . . . . .	350
7.51 SpringBasedElement Class Reference . . . . .	350
7.51.1 Detailed Description . . . . .	351
7.51.2 Constructor & Destructor Documentation . . . . .	351
7.51.2.1 __init__() . . . . .	351
7.51.3 Member Function Documentation . . . . .	353
7.51.3.1 CreateMember() . . . . .	353
7.51.3.2 Record() . . . . .	354
7.51.3.3 RecordNodeDef() . . . . .	354
7.51.3.4 Relnit() . . . . .	355
7.51.3.5 ShowInfo() . . . . .	355
7.51.3.6 UpdateStoredData() . . . . .	356
7.51.4 Member Data Documentation . . . . .	356
7.51.4.1 A . . . . .	357
7.51.4.2 data . . . . .	357
7.51.4.3 E . . . . .	357
7.51.4.4 ele_orientation . . . . .	357
7.51.4.5 element_array . . . . .	357

7.51.4.6 element_ID	357
7.51.4.7 geo_transf_ID	358
7.51.4.8 Initialized	358
7.51.4.9 iNode_ID	358
7.51.4.10 iNode_ID_spring	358
7.51.4.11 iSpring_ID	358
7.51.4.12 ly_mod	358
7.51.4.13 jNode_ID	359
7.51.4.14 jNode_ID_spring	359
7.51.4.15 jSpring_ID	359
7.51.4.16 mat_ID_i	359
7.51.4.17 mat_ID_j	359
7.51.4.18 section_name_tag	359
7.52 SpringBasedElementModifiedIMKSteelShape Class Reference	360
7.52.1 Detailed Description	360
7.52.2 Constructor & Destructor Documentation	360
7.52.2.1 __init__()	361
7.52.3 Member Data Documentation	362
7.52.3.1 section	362
7.52.3.2 section_name_tag	363
7.53 SpringBasedElementSteelShape Class Reference	363
7.53.1 Detailed Description	363
7.53.2 Constructor & Destructor Documentation	364
7.53.2.1 __init__()	364
7.53.3 Member Data Documentation	365
7.53.3.1 section	365
7.53.3.2 section_name_tag	365
7.54 SteelShape Class Reference	365
7.54.1 Detailed Description	367
7.54.2 Constructor & Destructor Documentation	367
7.54.2.1 __init__()	367
7.54.3 Member Function Documentation	369
7.54.3.1 Compute_iy()	369
7.54.3.2 Compute_iz()	369
7.54.3.3 ComputeA()	370
7.54.3.4 Computely()	370
7.54.3.5 Computelz()	371
7.54.3.6 ComputeWply()	371
7.54.3.7 ComputeWplz()	372
7.54.3.8 ReInit()	372
7.54.3.9 ShowInfo()	373
7.54.3.10 UpdateStoredData()	373

7.54.4 Member Data Documentation	374
7.54.4.1 A	374
7.54.4.2 bf	374
7.54.4.3 d	374
7.54.4.4 data	374
7.54.4.5 E	374
7.54.4.6 Fy	375
7.54.4.7 Fy_web	375
7.54.4.8 h_1	375
7.54.4.9 ly	375
7.54.4.10 iy	375
7.54.4.11 ly_mod	375
7.54.4.12 Iz	376
7.54.4.13 iz	376
7.54.4.14 L	376
7.54.4.15 My	376
7.54.4.16 n	376
7.54.4.17 name_tag	376
7.54.4.18 Npl	377
7.54.4.19 r	377
7.54.4.20 tf	377
7.54.4.21 tw	377
7.54.4.22 Type	377
7.54.4.23 Wply	377
7.54.4.24 Wplz	378
7.55 UnconfMander1988 Class Reference	378
7.55.1 Detailed Description	379
7.55.2 Constructor & Destructor Documentation	379
7.55.2.1 __init__()	379
7.55.3 Member Function Documentation	381
7.55.3.1 CheckApplicability()	381
7.55.3.2 Compute_ec()	381
7.55.3.3 Compute_ecp()	382
7.55.3.4 Compute_ecu()	382
7.55.3.5 Compute_et()	383
7.55.3.6 Compute_fct()	383
7.55.3.7 Concrete01()	384
7.55.3.8 Concrete04()	384
7.55.3.9 Relnit()	384
7.55.3.10 ShowInfo()	385
7.55.3.11 UpdateStoredData()	386
7.55.4 Member Data Documentation	386

7.55.4.1 beta	387
7.55.4.2 data	387
7.55.4.3 Ec	387
7.55.4.4 ec	387
7.55.4.5 ecp	387
7.55.4.6 ecu	387
7.55.4.7 et	388
7.55.4.8 fc	388
7.55.4.9 fct	388
7.55.4.10 ID	388
7.55.4.11 Initialized	388
7.55.4.12 section_name_tag	388
7.56 UnconfMander1988RCCircShape Class Reference	389
7.56.1 Detailed Description	389
7.56.2 Constructor & Destructor Documentation	389
7.56.2.1 __init__()	390
7.56.3 Member Data Documentation	390
7.56.3.1 section	391
7.56.3.2 section_name_tag	391
7.57 UnconfMander1988RCRectShape Class Reference	391
7.57.1 Detailed Description	391
7.57.2 Constructor & Destructor Documentation	392
7.57.2.1 __init__()	392
7.57.3 Member Data Documentation	393
7.57.3.1 section	393
7.57.3.2 section_name_tag	393
7.58 UniaxialBilinear Class Reference	393
7.58.1 Detailed Description	394
7.58.2 Constructor & Destructor Documentation	394
7.58.2.1 __init__()	394
7.58.3 Member Function Documentation	396
7.58.3.1 CheckApplicability()	396
7.58.3.2 ReInit()	397
7.58.3.3 ShowInfo()	397
7.58.3.4 Steel01()	398
7.58.3.5 UpdateStoredData()	399
7.58.4 Member Data Documentation	399
7.58.4.1 b	399
7.58.4.2 data	399
7.58.4.3 Ey	399
7.58.4.4 ey	400
7.58.4.5 fy	400

7.58.4.6 ID	400
7.58.4.7 Initialized	400
7.58.4.8 section_name_tag	400
7.59 UniaxialBilinearSteelShape Class Reference	401
7.59.1 Detailed Description	401
7.59.2 Constructor & Destructor Documentation	401
7.59.2.1 __init__()	402
7.59.3 Member Data Documentation	402
7.59.3.1 section	402
7.59.3.2 section_name_tag	402
7.60 UVC Class Reference	403
7.60.1 Detailed Description	403
7.60.2 Constructor & Destructor Documentation	404
7.60.2.1 __init__()	404
7.60.3 Member Function Documentation	405
7.60.3.1 CheckApplicability()	406
7.60.3.2 Relnit()	406
7.60.3.3 ShowInfo()	407
7.60.3.4 UpdateStoredData()	407
7.60.3.5 UVCuniaxial()	408
7.60.4 Member Data Documentation	408
7.60.4.1 a	408
7.60.4.2 b	408
7.60.4.3 cK	408
7.60.4.4 data	409
7.60.4.5 DInf	409
7.60.4.6 Ey	409
7.60.4.7 fy	409
7.60.4.8 gammaK	409
7.60.4.9 ID	409
7.60.4.10 Initialized	410
7.60.4.11 N	410
7.60.4.12 QInf	410
7.60.4.13 section_name_tag	410
7.61 UVCCalibrated Class Reference	410
7.61.1 Detailed Description	411
7.61.2 Constructor & Destructor Documentation	411
7.61.2.1 __init__()	411
7.61.3 'S355J2_25mm_plate'\n	412
7.61.4 'S355J2_50mm_plate'\n	412
7.61.5 'S355J2_HEB500_flange'\n	412
7.61.6 'S355J2_HEB500_web'\n	412

7.61.7 'S460NL_25mm_plate' \n	412
7.61.8 'S690QL_25mm_plate' \n	412
7.61.9 'A992Gr50_W14X82_web' \n	412
7.61.10 'A992Gr50_W14X82_flange' \n	412
7.61.11 'A500GrB_HSS305X16' \n	412
7.61.12 'BCP325_22mm_plate' \n	412
7.61.13 'BCR295_HSS350X22' \n	412
7.61.14 'HYP400_27mm_plate' \n	412
7.61.15 Member Data Documentation	413
7.61.15.1 calibration	413
7.62 UVCCalibratedRCCircShape Class Reference	414
7.62.1 Detailed Description	414
7.62.2 Constructor & Destructor Documentation	414
7.62.2.1 __init__()	415
7.62.3 Member Data Documentation	415
7.62.3.1 section	415
7.62.3.2 section_name_tag	415
7.63 UVCCalibratedRCRectShape Class Reference	416
7.63.1 Detailed Description	416
7.63.2 Constructor & Destructor Documentation	416
7.63.2.1 __init__()	417
7.63.3 Member Data Documentation	417
7.63.3.1 section	417
7.63.3.2 section_name_tag	417
7.64 UVCCalibratedSteelIShapeFlange Class Reference	418
7.64.1 Detailed Description	418
7.64.2 Constructor & Destructor Documentation	418
7.64.2.1 __init__()	419
7.64.3 Member Data Documentation	419
7.64.3.1 section	419
7.64.3.2 section_name_tag	419
7.65 UVCCalibratedSteelIShapeWeb Class Reference	420
7.65.1 Detailed Description	420
7.65.2 Constructor & Destructor Documentation	420
7.65.2.1 __init__()	421
7.65.3 Member Data Documentation	421
7.65.3.1 section	421
7.65.3.2 section_name_tag	421
7.66 WrongArgument Class Reference	422
7.66.1 Detailed Description	422
7.67 WrongDimension Class Reference	422
7.67.1 Detailed Description	422

7.68 WrongNodeIDConvention Class Reference . . . . .	423
7.68.1 Detailed Description . . . . .	423
7.68.2 Constructor & Destructor Documentation . . . . .	423
7.68.2.1 __init__() . . . . .	423
7.68.3 Member Data Documentation . . . . .	423
7.68.3.1 node . . . . .	424
7.69 ZeroDivision Class Reference . . . . .	424
7.69.1 Detailed Description . . . . .	424
7.70 ZeroLength Class Reference . . . . .	424
7.70.1 Detailed Description . . . . .	425
7.70.2 Constructor & Destructor Documentation . . . . .	425
7.70.2.1 __init__() . . . . .	425
7.70.3 Member Data Documentation . . . . .	425
7.70.3.1 element . . . . .	425
<b>8 File Documentation</b> . . . . .	<b>427</b>
8.1 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/AnalysisAndPostProcessing.py File Reference	427
8.2 AnalysisAndPostProcessing.py . . . . .	427
8.3 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Connections.py File Reference . . . . .	438
8.4 Connections.py . . . . .	438
8.5 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Constants.py File Reference . . . . .	439
8.6 Constants.py . . . . .	440
8.7 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/DataManagement.py File Reference . . . . .	440
8.8 DataManagement.py . . . . .	440
8.9 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py File Reference . . . . .	441
8.10 ErrorHandling.py . . . . .	442
8.11 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py File Reference . . . . .	443
8.12 Fibers.py . . . . .	443
8.13 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/FunctionalFeatures.py File Reference . . . . .	456
8.14 FunctionalFeatures.py . . . . .	456
8.15 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/GeometryTemplate.py File Reference . . . . .	462
8.16 GeometryTemplate.py . . . . .	463
8.17 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py File Reference . . . . .	466
8.18 MaterialModels.py . . . . .	468
8.19 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py File Reference . . . . .	509
8.20 MemberModel.py . . . . .	511
8.21 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/README.md File Reference . . . . .	533
8.22 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py File Reference . . . . .	533
8.23 Section.py . . . . .	533
8.24 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Units.py File Reference . . . . .	543
8.25 Units.py . . . . .	544
<b>Index</b> . . . . .	<b>547</b>





## Chapter 1

# OpenSeesPyAssistant

Help the use of OpenSeesPy tools and the implementation of material models, elements, fibers and much more.



## Chapter 2

# Namespace Index

### 2.1 Packages

Here are the packages with brief descriptions (if available):

<a href="#">AnalysisAndPostProcessing</a>	
Module with pre-made analysis and postprocessing functions . . . . .	13
<a href="#">Connections</a>	
Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs) . . . . .	13
<a href="#">Constants</a>	
Module with the values of a set of essential constants . . . . .	16
<a href="#">DataManagement</a>	
Module with the parent abstract class DataManagement . . . . .	18
<a href="#">ErrorHandling</a>	
Module dedicated to the error handling . . . . .	18
<a href="#">Fibers</a>	
Module for the fibers (rectangular, circular and I shape) . . . . .	19
<a href="#">FunctionalFeatures</a>	
Module with useful functions (discretise curves, ID conventions, etc)	
Carmine Schipani, 2021 . . . . .	24
<a href="#">GeometryTemplate</a>	
Module with geometry templates (nodes and/or elements with associated fibers, material models, etc) . . . . .	36
<a href="#">MaterialModels</a>	
Module for the material models . . . . .	43
<a href="#">MemberModel</a>	
Module for the member model . . . . .	50
<a href="#">Section</a>	
Module for the section (steel I shape profiles, RC circular/square/rectangular sections) . . . . .	54
<a href="#">Units</a>	
Module with the units conversion and the definition of the units used as default (m, N, s) . . . . .	56



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Analysis . . . . .	67
DataManagement	
Fibers . . . . .	140
FibersCirc . . . . .	141
FibersCircRCCircShape . . . . .	150
FibersIShape . . . . .	152
FibersIShapeSteelIShape . . . . .	161
FibersRect . . . . .	164
FibersRectRCRectShape . . . . .	174
MaterialModels . . . . .	248
ConfMander1988Circ . . . . .	84
ConfMander1988CircRCCircShape . . . . .	101
ConfMander1988Rect . . . . .	103
ConfMander1988RectRCRectShape . . . . .	125
GMP1970 . . . . .	217
GMP1970RCRectShape . . . . .	225
Gupta1999 . . . . .	228
Gupta1999SteelIShape . . . . .	241
ModifiedIMK . . . . .	252
ModifiedIMKSteelIShape . . . . .	273
Skiadopoulos2021 . . . . .	326
Skiadopoulos2021RCS . . . . .	344
Skiadopoulos2021SteelIShape . . . . .	347
UVC . . . . .	403
UVCCalibrated . . . . .	410
UVCCalibratedRCCircShape . . . . .	414
UVCCalibratedRCRectShape . . . . .	416
UVCCalibratedSteelIShapeFlange . . . . .	418
UVCCalibratedSteelIShapeWeb . . . . .	420
UnconfMander1988 . . . . .	378
UnconfMander1988RCCircShape . . . . .	389
UnconfMander1988RCRectShape . . . . .	391
UniaxialBilinear . . . . .	393
UniaxialBilinearSteelIShape . . . . .	401

MemberModel . . . . .	250
ElasticElement . . . . .	130
ElasticElementSteelShape . . . . .	138
ForceBasedElement . . . . .	177
ForceBasedElementFibersCircRCCircShape . . . . .	185
ForceBasedElementFibersIShapeSteelShape . . . . .	188
ForceBasedElementFibersRectRCRectShape . . . . .	191
GIFBElement . . . . .	193
GIFBElementFibersCircRCCircShape . . . . .	205
GIFBElementFibersRectRCRectShape . . . . .	208
GIFBElementRCCircShape . . . . .	211
GIFBElementRCRectShape . . . . .	214
PanelZone . . . . .	277
PanelZoneRCS . . . . .	286
PanelZoneSteelShape . . . . .	289
PanelZoneSteelShapeGupta1999 . . . . .	292
PanelZoneSteelShapeSkiadopoulos2021 . . . . .	294
SpringBasedElement . . . . .	350
SpringBasedElementModifiedIMKSteelShape . . . . .	360
SpringBasedElementSteelShape . . . . .	363
Section . . . . .	325
RCCircShape . . . . .	297
RCRectShape . . . . .	308
RCSquareShape . . . . .	323
SteelShape . . . . .	365
Exception . . . . .	
InconsistentGeometry . . . . .	247
MemberFailure . . . . .	249
NegativeValue . . . . .	276
NoApplicability . . . . .	276
PositiveValue . . . . .	296
WrongArgument . . . . .	422
WrongDimension . . . . .	422
WrongNodeIDConvention . . . . .	423
ZeroDivision . . . . .	424
ZeroLength . . . . .	424
IDGenerator . . . . .	244
ABC . . . . .	
DataManagement . . . . .	127

## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Analysis</a>	Class dedicated to the analysis of the OpenSeesPy model . . . . .	67
<a href="#">ConfMander1988Circ</a>	Class that stores functions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01 . . . . .	84
<a href="#">ConfMander1988CircRCCircShape</a>	Class that is the children of <a href="#">ConfMander1988Circ</a> and combine the class RCCircShape (section) to retrieve the information needed . . . . .	101
<a href="#">ConfMander1988Rect</a>	Class that stores functions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01 . . . . .	103
<a href="#">ConfMander1988RectRCRectShape</a>	Class that is the children of <a href="#">ConfMander1988Rect</a> and combine the class RCRectShape (section) to retrieve the information needed . . . . .	125
<a href="#">DataManagement</a>	Abstract parent class for data management . . . . .	127
<a href="#">ElasticElement</a>	Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model . . . . .	130
<a href="#">ElasticElementSteelShape</a>	Class that is the children of <a href="#">ElasticElement</a> and combine the class SteelShape (section) to retrieve the information needed . . . . .	138
<a href="#">Fibers</a>	Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model . . . . .	140
<a href="#">FibersCirc</a>	Class that stores functions, material properties, geometric and mechanical parameters for a circular RC fiber section . . . . .	141
<a href="#">FibersCircRCCircShape</a>	Class that is the children of <a href="#">FibersCirc</a> and combine the class RCCircShape (section) to retrieve the information needed . . . . .	150
<a href="#">FibersIShape</a>	Class that stores functions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section . . . . .	152

<a href="#">FibersIShapeSteelShape</a>	
Class that is the children of <a href="#">FibersIShape</a> and combine the class <a href="#">SteelShape</a> (section) to retrieve the information needed . . . . .	161
<a href="#">FibersRect</a>	
Class that stores functions, material properties, geometric and mechanical parameters for a rectangular RC fiber section . . . . .	164
<a href="#">FibersRectRCRectShape</a>	
Class that is the children of <a href="#">FibersRect</a> and combine the class <a href="#">RCRectShape</a> (section) to retrieve the information needed . . . . .	174
<a href="#">ForceBasedElement</a>	
Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model . . . . .	177
<a href="#">ForceBasedElementFibersCircRCCircShape</a>	
Class that is the children of <a href="#">ForceBasedElement</a> and combine the class <a href="#">FibersCircRCCircShape</a> (fiber section) to retrieve the information needed . . . . .	185
<a href="#">ForceBasedElementFibersIShapeSteelShape</a>	
Class that is the children of <a href="#">ForceBasedElement</a> and combine the class <a href="#">FibersIShapeSteelShape</a> (fiber section) to retrieve the information needed . . . . .	188
<a href="#">ForceBasedElementFibersRectRCRectShape</a>	
Class that is the children of <a href="#">ForceBasedElement</a> and combine the class <a href="#">FibersRectRCRectShape</a> (fiber section) to retrieve the information needed . . . . .	191
<a href="#">GIFBElement</a>	
Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model . . . . .	193
<a href="#">GIFBElementFibersCircRCCircShape</a>	
Class that is the children of <a href="#">GIFBElement</a> and combine the class <a href="#">FibersCircRCCircShape</a> (fiber section) to retrieve the information needed . . . . .	205
<a href="#">GIFBElementFibersRectRCRectShape</a>	
Class that is the children of <a href="#">GIFBElement</a> and combine the class <a href="#">FibersRectRCRectShape</a> (fiber section) to retrieve the information needed . . . . .	208
<a href="#">GIFBElementRCCircShape</a>	
Class that is the children of <a href="#">GIFBElement</a> and combine the class <a href="#">RCCircShape</a> (section) to retrieve the information needed . . . . .	211
<a href="#">GIFBElementRCRectShape</a>	
Class that is the children of <a href="#">GIFBElement</a> and combine the class <a href="#">RCRectShape</a> (section) to retrieve the information needed . . . . .	214
<a href="#">GMP1970</a>	
Class that stores functions and material properties of the vertical steel reinforcement bars with Giuffr�, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is <a href="#">Steel02</a> . . . . .	217
<a href="#">GMP1970RCRectShape</a>	
Class that is the children of <a href="#">GMP1970</a> and combine the class <a href="#">RCRectShape</a> (section) to retrieve the information needed . . . . .	225
<a href="#">Gupta1999</a>	
Class that stores functions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is <a href="#">Hysteresis</a> . . . . .	228
<a href="#">Gupta1999SteelShape</a>	
Class that is the children of <a href="#">Gupta1999</a> and combine the class <a href="#">SteelShape</a> (section) to retrieve the information needed . . . . .	241
<a href="#">IDGenerator</a>	
Class that manage the ID generation . . . . .	244
<a href="#">InconsistentGeometry</a>	
Exception class for the "inconsistent geometry" error . . . . .	247
<a href="#">MaterialModels</a>	
Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model . . . . .	248



<a href="#">MemberFailure</a>	Exception class for the "member failure" error . . . . .	249
<a href="#">MemberModel</a>	Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model . . . . .	250
<a href="#">ModifiedIMK</a>	Class that stores funcions and material properties of a steel double symmetric I-shape profile with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin . . . . .	252
<a href="#">ModifiedIMKSteelShape</a>	Class that is the children of <a href="#">ModifiedIMK</a> and combine the class SteelShape (section) to retrieve the information needed . . . . .	273
<a href="#">NegativeValue</a>	Exception class for the "negative value (argument or result)" error . . . . .	276
<a href="#">NoApplicability</a>	Exception class for the "no applicability of formula of theory" error . . . . .	276
<a href="#">PanelZone</a>	Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model . . . . .	277
<a href="#">PanelZoneRCS</a>	WIP: Class that is the children of <a href="#">PanelZone</a> and it's used for the panel zone in a RCS (RC column continous, Steel beam) . . . . .	286
<a href="#">PanelZoneSteelShape</a>	Class that is the children of <a href="#">PanelZone</a> and combine the class SteelShape (section) to retrieve the information needed . . . . .	289
<a href="#">PanelZoneSteelShapeGupta1999</a>	Class that is the children of <a href="#">PanelZoneSteelShape</a> and automatically create the spring material model Gupta 1999 (ID = master_node_ID) . . . . .	292
<a href="#">PanelZoneSteelShapeSkiadopoulos2021</a>	Class that is the children of <a href="#">PanelZoneSteelShape</a> and automatically create the spring material model Skiadopoulos 2021 (ID = master_node_ID) . . . . .	294
<a href="#">PositiveValue</a>	Exception class for the "positive value (argument or result)" error . . . . .	296
<a href="#">RCCircShape</a>	Class that stores funcions, geometric and mechanical properties of RC circular shape profile . . . . .	297
<a href="#">RCRectShape</a>	Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile . . . . .	308
<a href="#">RCSquareShape</a>	Class that is the children of <a href="#">RCRectShape</a> and cover the specific case of square RC sections . . . . .	323
<a href="#">Section</a>	Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc) . . . . .	325
<a href="#">Skiadopoulos2021</a>	Class that stores funcions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis . . . . .	326
<a href="#">Skiadopoulos2021RCS</a>	WIP: Class that is the children of <a href="#">Skiadopoulos2021</a> and it's used for the panel zone spring in a RCS (RC column continous, Steel beam) . . . . .	344
<a href="#">Skiadopoulos2021SteelShape</a>	Class that is the children of <a href="#">Skiadopoulos2021</a> and combine the class SteelShape (section) to retrieve the information needed . . . . .	347
<a href="#">SpringBasedElement</a>	Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model . . . . .	350
<a href="#">SpringBasedElementModifiedIMKSteelShape</a>	Class that is the children of <a href="#">SpringBasedElement</a> and combine the class SteelShape (section) to retrieve the information needed . . . . .	360

<a href="#">SpringBasedElementSteelShape</a>	
Class that is the children of <a href="#">SpringBasedElement</a> and combine the class <a href="#">SteelShape</a> (section) to retrieve the information needed . . . . .	363
<a href="#">SteelShape</a>	
Class that stores funcions, geometric and mechanical properties of a steel double symmetric I-shape profile . . . . .	365
<a href="#">UnconfMander1988</a>	
Class that stores funcions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01 . . . . .	378
<a href="#">UnconfMander1988RCCircShape</a>	
Class that is the children of <a href="#">UnconfMander1988</a> and combine the class <a href="#">RCCircShape</a> (section) to retrieve the information needed . . . . .	389
<a href="#">UnconfMander1988RCRectShape</a>	
Class that is the children of <a href="#">UnconfMander1988</a> and combine the class <a href="#">RCRectShape</a> (section) to retrieve the information needed . . . . .	391
<a href="#">UniaxialBilinear</a>	
Class that stores funcions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01 . . . . .	393
<a href="#">UniaxialBilinearSteelShape</a>	
Class that is the children of <a href="#">UniaxialBilinear</a> and combine the class <a href="#">SteelShape</a> (section) to retrieve the information needed . . . . .	401
<a href="#">UVC</a>	
Class that stores funcions and material properties of a steel profile or reinforcing bar with Up-dated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial . . . . .	403
<a href="#">UVCCalibrated</a>	
Class that is the children of <a href="#">UVC</a> that retrieve calibrated parameters from UVC_calibrated_↔ parameters.txt . . . . .	410
<a href="#">UVCCalibratedRCCircShape</a>	
Class that is the children of <a href="#">UVCCalibrated</a> and combine the class <a href="#">RCCircShape</a> (section) to retrieve the information needed . . . . .	414
<a href="#">UVCCalibratedRCRectShape</a>	
Class that is the children of <a href="#">UVCCalibrated</a> and combines the class <a href="#">RCRectShape</a> (section) to retrieve the information needed . . . . .	416
<a href="#">UVCCalibratedSteelShapeFlange</a>	
Class that is the children of <a href="#">UVCCalibrated</a> and combine the class <a href="#">SteelShape</a> (section) to retrieve the information needed for the material model of the flange (often used fo the entire section) . . . . .	418
<a href="#">UVCCalibratedSteelShapeWeb</a>	
Class that is the children of <a href="#">UVCCalibrated</a> and combine the class <a href="#">SteelShape</a> (section) to retrieve the information needed for the material model of the web . . . . .	420
<a href="#">WrongArgument</a>	
Exception class for the "input of a wrong argument" error . . . . .	422
<a href="#">WrongDimension</a>	
Exception class for the "wrong array dimensions" error . . . . .	422
<a href="#">WrongNodeIDConvention</a>	
Exception class for the "wrong node ID convention definition" error . . . . .	423
<a href="#">ZeroDivision</a>	
Exception class for the "zero division" error . . . . .	424
<a href="#">ZeroLength</a>	
Exception class for the "zero length element (non intentional)" error . . . . .	424

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">AnalysisAndPostProcessing.py</a> . . . . .	427
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">Connections.py</a> . . . . .	438
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">Constants.py</a> . . . . .	439
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">DataManagement.py</a> . . . . .	440
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">ErrorHandling.py</a> . . . . .	441
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">Fibers.py</a> . . . . .	443
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">FunctionalFeatures.py</a> . . . . .	456
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">GeometryTemplate.py</a> . . . . .	462
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">MaterialModels.py</a> . . . . .	466
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">MemberModel.py</a> . . . . .	509
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">Section.py</a> . . . . .	533
/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ <a href="#">Units.py</a> . . . . .	543



## Chapter 6

# Namespace Documentation

### 6.1 AnalysisAndPostProcessing Namespace Reference

Module with pre-made analysis and postprocessing functions.

#### Classes

- class [Analysis](#)  
*Class dedicated to the analysis of the OpenSeesPy model.*

#### 6.1.1 Detailed Description

Module with pre-made analysis and postprocessing functions.

Carmine Schipani, 2021

### 6.2 Connections Namespace Reference

Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs).

#### Functions

- def [Pin](#) (int NodeRID, int NodeCID)  
*Function that constrains the translational DOF with a multi-point constraint.*
- def [RigidSupport](#) (int NodeID)  
*Function that fixes the x, y movements and the rotation of one node.*
- def [RotationalSpring](#) (int ElementID, int NodeRID, int NodeCID, int MatID, Rigid=False)  
*Function that defines a zero-length spring and constrains the translations DOFs of the spring.*

## 6.2.1 Detailed Description

Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs).

Carmine Schipani, 2021

## 6.2.2 Function Documentation

### 6.2.2.1 Pin()

```
def Connections.Pin (
    int NodeRID,
    int NodeCID )
```

Function that constrains the translational DOF with a multi-point constraint.

#### Parameters

<i>NodeRID</i>	(int): Node ID which will be retained by the multi-point constraint
<i>NodeCID</i>	(int): Node ID which will be constrained by the multi-point constraint

#### Exceptions

<i>WrongArgument</i>	The IDs passed needs to be different.
<i>NegativeValue</i>	The ID of NodeRID needs to be a positive integer.
<i>NegativeValue</i>	The ID of NodeCID needs to be a positive integer.

Definition at line 22 of file [Connections.py](#).

```
00022 def Pin(NodeRID: int, NodeCID: int):
00023     """
00024     Function that constrains the translational DOF with a multi-point constraint.
00025
00026     @param NodeRID (int): Node ID which will be retained by the multi-point constraint
00027     @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
00028
00029     @exception WrongArgument: The IDs passed needs to be different.
00030     @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
00031     @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
00032     """
00033     if NodeCID == NodeRID: raise WrongArgument()
00034     if NodeRID < 1: raise NegativeValue()
00035     if NodeCID < 1: raise NegativeValue()
00036
00037     # Constrain the translational DOF with a multi-point constraint
00038     # retained constrained DOF_1 DOF_2
00039     equalDOF(NodeRID, NodeCID, 1, 2)
00040
00041
```

### 6.2.2.2 RigidSupport()

```
def Connections.RigidSupport (
    int NodeID )
```

Function that fixes the x, y movements and the rotation of one node.

#### Parameters

<i>NodeID</i>	(int): ID of the node to be fixed
---------------	-----------------------------------

#### Exceptions

<i>NegativeValue</i>	The ID of NodeID needs to be a positive integer.
----------------------	--

Definition at line 9 of file [Connections.py](#).

```

00009 def RigidSupport(NodeID: int):
00010     """
00011     Function that fixes the x, y movements and the rotation of one node.
00012
00013     @param NodeID (int): ID of the node to be fixed
00014
00015     @exception NegativeValue: The ID of NodeID needs to be a positive integer.
00016     """
00017     if NodeID < 1: raise NegativeValue()
00018
00019     fix(NodeID, 1, 1, 1)
00020
00021

```

### 6.2.2.3 RotationalSpring()

```

def Connections.RotationalSpring (
    int ElementID,
    int NodeRID,
    int NodeCID,
    int MatID,
    Rigid = False )

```

Function that defines a zero-length spring and constrains the translations DOFs of the spring.

Can be used also to create rigid connections.

#### Parameters

<i>ElementID</i>	(int): ID of the zerolength element that models the spring
<i>NodeRID</i>	(int): Node ID which will be retained by the multi-point constraint
<i>NodeCID</i>	(int): Node ID which will be constrained by the multi-point constraint
<i>MatID</i>	(int): ID of the material model chosen
<i>Rigid</i>	(bool, optional): Optional argument that transforms the joint in a completely rigid connection. Defaults to False.

#### Exceptions

<i>NegativeValue</i>	The ID of ElementID needs to be a positive integer.
<i>NegativeValue</i>	The ID of NodeCID needs to be a positive integer.
<i>NegativeValue</i>	The ID of NodeRID needs to be a positive integer.
<i>WrongArgument</i>	The IDs of the nodes passed needs to be different.

## Exceptions

<i>NegativeValue</i>	The ID of MatID needs to be a positive integer.
----------------------	---

Definition at line 42 of file [Connections.py](#).

```

00042 def RotationalSpring(ElementID: int, NodeRID: int, NodeCID: int, MatID: int, Rigid = False):
00043     """
00044     Function that defines a zero-length spring and constrains the translations DOFs of the spring. Can
        be used also to create rigid connections.
00045
00046     @param ElementID (int): ID of the zerolength element that models the spring
00047     @param NodeRID (int): Node ID which will be retained by the multi-point constraint
00048     @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
00049     @param MatID (int): ID of the material model chosen
00050     @param Rigid (bool, optional): Optional argument that transforms the joint in a completely rigid
        connection. Defaults to False.
00051
00052     @exception NegativeValue: The ID of ElementID needs to be a positive integer.
00053     @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
00054     @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
00055     @exception WrongArgument: The IDs of the nodes passed needs to be different.
00056     @exception NegativeValue: The ID of MatID needs to be a positive integer.
00057     """
00058     if ElementID < 1: raise NegativeValue()
00059     if NodeCID < 1: raise NegativeValue()
00060     if NodeRID < 1: raise NegativeValue()
00061     if NodeCID == NodeRID: raise WrongArgument()
00062     if MatID < 1: raise NegativeValue()
00063
00064     if not Rigid:
00065         # Zero length element (spring)
00066         element("zeroLength", ElementID, NodeRID, NodeCID, "-mat", MatID, "-dir", 6)
00067
00068         # Constrain the translational DOF with a multi-point constraint
00069         Pin(NodeRID, NodeCID)
00070     else:
00071         equalDOF(NodeRID, NodeCID, 1, 2, 3)
00072
00073
00074
00075

```

## 6.3 Constants Namespace Reference

Module with the values of a set of essential constants.

### Variables

- float [G\\_CONST](#) = 9.810\*m\_unit/s\_unit\*\*2
- int [MAX\\_ITER](#) = 100
- int [MAX\\_ITER\\_INTEGRATION](#) = 50
- float [RIGID](#) = 100.0
- float [TOL](#) = 1.0e-6
- float [TOL\\_INTEGRATION](#) = 1.0e-12
- float [ZERO](#) = 1.0e-9

### 6.3.1 Detailed Description

Module with the values of a set of essential constants.

They are consistent with the units defined in [Units](#).

Carmine Schipani, 2021



## 6.3.2 Variable Documentation

### 6.3.2.1 G\_CONST

```
float G_CONST = 9.810*m_unit/s_unit**2
```

Definition at line 11 of file [Constants.py](#).

### 6.3.2.2 MAX\_ITER

```
int MAX_ITER = 100
```

Definition at line 13 of file [Constants.py](#).

### 6.3.2.3 MAX\_ITER\_INTEGRATION

```
int MAX_ITER_INTEGRATION = 50
```

Definition at line 14 of file [Constants.py](#).

### 6.3.2.4 RIGID

```
float RIGID = 100.0
```

Definition at line 12 of file [Constants.py](#).

### 6.3.2.5 TOL

```
float TOL = 1.0e-6
```

Definition at line 8 of file [Constants.py](#).

### 6.3.2.6 TOL\_INTEGRATION

```
float TOL_INTEGRATION = 1.0e-12
```

Definition at line 9 of file [Constants.py](#).

### 6.3.2.7 ZERO

```
float ZERO = 1.0e-9
```

Definition at line 10 of file [Constants.py](#).

## 6.4 DataManagement Namespace Reference

Module with the parent abstract class DataManagement.

### Classes

- class [DataManagement](#)  
*Abstract parent class for data management.*

### 6.4.1 Detailed Description

Module with the parent abstract class DataManagement.

Carmine Schipani, 2021

## 6.5 ErrorHandler Namespace Reference

Module dedicated to the error handling.

## Classes

- class [InconsistentGeometry](#)  
*Exception class for the "inconsistent geometry" error.*
- class [MemberFailure](#)  
*Exception class for the "member failure" error.*
- class [NegativeValue](#)  
*Exception class for the "negative value (argument or result)" error.*
- class [NoApplicability](#)  
*Exception class for the "no applicability of formula of theory" error.*
- class [PositiveValue](#)  
*Exception class for the "positive value (argument or result)" error.*
- class [WrongArgument](#)  
*Exception class for the "input of a wrong argument" error.*
- class [WrongDimension](#)  
*Exception class for the "wrong array dimensions" error.*
- class [WrongNodeIDConvention](#)  
*Exception class for the "wrong node ID convention definition" error.*
- class [ZeroDivision](#)  
*Exception class for the "zero division" error.*
- class [ZeroLength](#)  
*Exception class for the "zero length element (non intentional)" error.*

### 6.5.1 Detailed Description

Module dedicated to the error handling.

Carmine Schipani, 2021

## 6.6 Fibers Namespace Reference

Module for the fibers (rectangular, circular and I shape).

## Classes

- class [Fibers](#)  
*Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.*
- class [FibersCirc](#)  
*Class that stores functions, material properties, geometric and mechanical parameters for a circular RC fiber section.*
- class [FibersCircRCCircShape](#)  
*Class that is the children of [FibersCirc](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.*
- class [FibersIShape](#)  
*Class that stores functions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.*
- class [FibersIShapeSteelIShape](#)

Class that is the children of [FibersIShape](#) and combine the class [SteelIShape](#) (section) to retrieve the information needed.

- class [FibersRect](#)

Class that stores functions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.

- class [FibersRectRCRectShape](#)

Class that is the children of [FibersRect](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.

## Functions

- def [create\\_fiber\\_section](#) (fiber\_info)  
*Initialise fiber cross-section with OpenSeesPy commands.*
- def [plot\\_fiber\\_section](#) (fiber\_info, fill\_shapes=True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'])  
*Plot fiber cross-section.*

### 6.6.1 Detailed Description

Module for the fibers (rectangular, circular and I shape).

Carmine Schipani, 2021

### 6.6.2 Function Documentation

#### 6.6.2.1 [create\\_fiber\\_section\(\)](#)

```
def Fibers.create_fiber_section (
    fiber_info )
```

Initialise fiber cross-section with OpenSeesPy commands.

For examples, see [plot\\_fiber\\_section](#). Inspired by [fib\\_sec\\_list\\_to\\_cmds](#) from [ops\\_vis](#) written by Seweryn Kokot

#### Parameters

<i>fiber_info</i>	<p>(list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section:</p> <pre>['section', 'Fiber', ID, '-GJ', GJ]</pre> <p>The other lists have one of the following format (coordinate input: (y, z)!):</p> <pre>['layer', 'bar', mat_ID, A, y, z] # one bar</pre> <pre>['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar)</pre> <pre>['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius)</pre> <pre>['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2)</pre> <pre>['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L)</pre> <pre>['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri = internal radius, re = external radius)</pre>
-------------------	---

Definition at line 857 of file [Fibers.py](#).

```

00857 def create_fiber_section(fiber_info):
00858     """
00859     Initialise fiber cross-section with OpenSeesPy commands.
00860     For examples, see plot_fiber_section.
00861     Inspired by fib_sec_list_to_cmds from ops_vis written by Seweryn Kokot
00862
00863     @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first
00864     list defines the fiber section: \n
00865     ['section', 'Fiber', ID, '-GJ', GJ] \n
00866     The other lists have one of the following format (coordinate input: (y, z)!): \n
00867     ['layer', 'bar', mat_ID, A, y, z] # one bar \n
00868     ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first
00869     bar, J = last bar) \n
00870     ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars
00871     (with C = center, r = radius) \n
00872     ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK =
00873     b/2) \n
00874     ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped
00875     (starting from bottom left, counterclockwise: I, J, K, L) \n
00876     ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri =
00877     internal radius, re = external radius)
00878     """
00879     for dat in fiber_info:
00880         if dat[0] == 'section':
00881             fib_ID, GJ = dat[2], dat[4]
00882             section('Fiber', fib_ID, 'GJ', GJ)
00883
00884         if dat[0] == 'layer':
00885             mat_ID = dat[2]
00886             if dat[1] == 'straight':
00887                 n_bars = dat[3]
00888                 As = dat[4]
00889                 Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
00890                 layer('straight', mat_ID, n_bars, As, Iy, Iz, Jy, Jz)
00891             if dat[1] == 'bar':
00892                 As = dat[3]
00893                 Iy = dat[4]
00894                 Iz = dat[5]
00895                 fiber(Iy, Iz, As, mat_ID)
00896                 # layer('straight', mat_ID, 1, As, Iy, Iz, Iy, Iz)
00897             if dat[1] == 'circ':
00898                 n_bars, As = dat[3], dat[4]
00899                 yC, zC, r = dat[5], dat[6], dat[7]
00900                 if len(dat) > 8:
00901                     a0_deg = dat[8]
00902                 else:
00903                     a0_deg = 0.
00904                 a1_deg = 360. - 360./n_bars + a0_deg
00905                 if len(dat) > 9: a1_deg = dat[9]
00906                 layer('circ', mat_ID, n_bars, As, yC, zC, r, a0_deg, a1_deg)
00907
00908         if dat[0] == 'patch':
00909             mat_ID = dat[2]
00910             nIJ = dat[4]
00911             nJK = dat[3]
00912
00913             if dat[1] == 'quad' or dat[1] == 'quadr':
00914                 Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
00915                 Ky, Kz, Ly, Lz = dat[9], dat[10], dat[11], dat[12]
00916                 patch('quad', mat_ID, nIJ, nJK, Iy, Iz, Jy, Jz, Ky, Kz,
00917                     Ly, Lz)
00918
00919             if dat[1] == 'rect':
00920                 Iy, Iz, Ky, Kz = dat[5], dat[6], dat[7], dat[8]
00921                 patch('rect', mat_ID, nIJ, nJK, Iy, Iz, Ky, Kz)
00922                 # patch('rect', mat_ID, nIJ, nJK, Iy, Kz, Ky, Iz)
00923
00924             if dat[1] == 'circ':
00925                 mat_ID, nc, nr = dat[2], dat[3], dat[4]
00926                 yC, zC, ri, re = dat[5], dat[6], dat[7], dat[8]
00927                 if len(dat) > 9:
00928                     a0 = dat[9]
00929                 else:
00930                     a0 = 0.
00931                 a1 = 360. + a0
00932                 if len(dat) > 10: a1 = dat[10]
00933                 patch('circ', mat_ID, nc, nr, yC, zC, ri, re, a0, a1)
00934

```

### 6.6.2.2 plot\_fiber\_section()

```
def Fibers.plot_fiber_section (
    fiber_info,
    fill_shapes = True,
    matcolor = ['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'] )
```

Plot fiber cross-section.

Coordinate system used: plotting coordinte = (x, y), fiber section coordinate (z, y) = (-x, y)  
Inspired by plot\_fiber\_section from ops\_vis written by Seweryn Kokot.

#### Parameters

<i>fiber_info</i>	(list): List of lists (be careful with the local coordinate system!). The first list defines the fiber section: [ <i>'section'</i> , <i>'Fiber'</i> , ID, <i>'-GJ'</i> , GJ] The other lists have one of the following format (coordinate input: (y, z)!): [ <i>'layer'</i> , <i>'bar'</i> , mat_ID, A, y, z] # one bar [ <i>'layer'</i> , <i>'straight'</i> , mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first bar, J = last bar) [ <i>'layer'</i> , <i>'circ'</i> , mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars (with C = center, r = radius) [ <i>'patch'</i> , <i>'rect'</i> , mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK = b/2) [ <i>'patch'</i> , <i>'quad'</i> , mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped (starting from bottom left, counterclockwise: I, J, K, L) [ <i>'patch'</i> , <i>'circ'</i> , mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri = internal radius, re = external radius)
<i>fill_shapes</i>	(bool, optional): Option to fill fibers with color specified in matcolor. Defaults to True.
<i>matcolor</i>	(list, optional): List of colors for various material IDs. Defaults to [ <i>'#808080'</i> , <i>'#D3D3D3'</i> , <i>'r'</i> , <i>'b'</i> , <i>'g'</i> , <i>'y'</i> ].

Example 1: Simple rectangle with 2 rebars (D = diameter) on top (e distance from the top and from the lateral borders). Rectangle with first corner = I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2) fib\_sec = [*'section'*, *'Fiber'*, ID, *'-GJ'*, GJ], [*'patch'*, *'rect'*, concrete\_mat\_ID, \*discr, yI, zI, yK, zK], [*'layer'*, *'bar'*, bars\_mat\_ID, Ay, yI-e-D/2, zI-e-D/2], # left rebar [*'layer'*, *'bar'*, bars\_mat\_ID, Ay, yI-e-D/2, -(zI-e-D/2)]] # right rebar

Example 2: double symmetric I shape. Each rectangle (2 flanges and 1 web): first corner = I (bottom right) and second corner = K (top left); number of fibers = discr (list of 2) fib\_sec = [*'section'*, *'Fiber'*, ID, *'-GJ'*, GJ], [*'patch'*, *'rect'*, mat\_ID, \*discr, yI\_tf, zI\_tf, yK\_tf, zK\_tf], # top flange [*'patch'*, *'rect'*, mat\_ID, \*discr, yI\_bf, zI\_bf, yK\_bf, zK\_bf], # bottom flange [*'patch'*, *'rect'*, mat\_ID, \*discr, yI\_w, zI\_w, yK\_w, zK\_w]] # web

Definition at line 673 of file [Fibers.py](#).

```
00673 def plot_fiber_section(fiber_info, fill_shapes = True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g',
00674     'y']):
00675     """
00676     Plot fiber cross-section. Coordinate system used: plotting coordinte = (x, y), fiber section
00677     coordinate (z, y) = (-x, y)
00678     Inspired by plot_fiber_section from ops_vis written by Seweryn Kokot.
00679
00678     @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first
00679     list defines the fiber section: \n
00679     ['section', 'Fiber', ID, '-GJ', GJ] \n
00680     The other lists have one of the following format (coordinate input: (y, z)!): \n
00681     ['layer', 'bar', mat_ID, A, y, z] # one bar \n
00682     ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first
00683     bar, J = last bar) \n
00683     ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars
00684     (with C = center, r = radius) \n
00684     ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK =
00685     b/2) \n
00685     ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped
00685     (starting from bottom left, counterclockwise: I, J, K, L) \n
```

```

00686     ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri =
internal radius, re = external radius)
00687     @param fill_shapes (bool, optional): Option to fill fibers with color specified in matcolor.
Defaults to True.
00688     @param matcolor (list, optional): List of colors for various material IDs. Defaults to ['#808080',
'#D3D3D3', 'r', 'b', 'g', 'y'].
00689
00690     Example 1: Simple rectangle with 2 rebars (D = diameter) on top (e distance from the top and from
the lateral borders).
00691     Rectangle with first corner = I (bottom right) and second corner = K (top left); number of
fibers = discr (list of 2)
00692     fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
00693               ['patch', 'rect', concrete_mat_ID, *discr, yI, zI, yK, zK],
00694               ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, zI-e-D/2], # left rebar
00695               ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, -(zI-e-D/2)]] # right rebar
00696
00697     Example 2: double symmetric I shape.
00698     Each rectangle (2 flanges and 1 web): first corner = I (bottom right) and second corner = K
(top left); number of fibers = discr (list of 2)
00699     fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
00700               ['patch', 'rect', mat_ID, *discr, yI_tf, zI_tf, yK_tf, zK_tf], # top flange
00701               ['patch', 'rect', mat_ID, *discr, yI_bf, zI_bf, yK_bf, zK_bf], # bottom flange
00702               ['patch', 'rect', mat_ID, *discr, yI_w, zI_w, yK_w, zK_w]] # web
00703
00704
00705     mat_to_col = {}
00706     fig, ax = plt.subplots()
00707     ax.grid(False)
00708
00709     for item in fiber_info:
00710         if item[0] == 'section':
00711             fib_ID = item[2]
00712         if item[0] == 'layer':
00713             matID = item[2]
00714             mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
00715             if item[1] == 'bar':
00716                 As = item[3]
00717                 Iy = item[4]
00718                 Iz = item[5]
00719                 r = np.sqrt(As / np.pi)
00720                 bar = Circle((-Iz, Iy), r, ec='k', fc='k', zorder=10)
00721                 ax.add_patch(bar)
00722             if item[1] == 'straight':
00723                 n_bars = item[3]
00724                 As = item[4]
00725                 Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
00726                 r = np.sqrt(As / np.pi)
00727                 Y = np.linspace(Iy, Jy, n_bars)
00728                 Z = np.linspace(Iz, Jz, n_bars)
00729                 for zi, yi in zip(Z, Y):
00730                     bar = Circle((-zi, yi), r, ec='k', fc=mat_to_col[matID], zorder=10)
00731                     ax.add_patch(bar)
00732             if item[1] == 'circ':
00733                 n_bars, As = item[3], item[4]
00734                 yC, zC, r = item[5], item[6], item[7]
00735                 if len(item) > 8:
00736                     a0_deg = item[8]
00737                 else:
00738                     a0_deg = 0.
00739                 a1_deg = 360. - 360./n_bars + a0_deg
00740                 if len(item) > 9: a1_deg = item[9]
00741
00742                 a0_rad, a1_rad = np.pi * a0_deg / 180., np.pi * a1_deg / 180.
00743                 r_bar = np.sqrt(As / np.pi)
00744                 thetas = np.linspace(a0_rad, a1_rad, n_bars)
00745                 Y = yC + r * np.cos(thetas)
00746                 Z = zC + r * np.sin(thetas)
00747                 for zi, yi in zip(Z, Y):
00748                     bar = Circle((-zi, yi), r_bar, ec='k', fc=mat_to_col[matID], zorder=10)
00749                     ax.add_patch(bar)
00750
00751             if (item[0] == 'patch' and (item[1] == 'quad' or item[1] == 'quadr' or
00752                                     item[1] == 'rect')):
00753                 matID, nIJ, nJK = item[2], item[3], item[4]
00754                 mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
00755
00756             if item[1] == 'quad' or item[1] == 'quadr':
00757                 Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
00758                 Ky, Kz, Ly, Lz = item[9], item[10], item[11], item[12]
00759
00760             if item[1] == 'rect':
00761                 Iy, Iz, Ky, Kz = item[5], item[6], item[7], item[8]
00762                 Jy, Jz, Ly, Lz = Iy, Kz, Ky, Iz
00763                 # check order of definition
00764                 if Kz-Iz < 0 or Ky-Iy < 0: print("!!!!!! WARNING !!!!!!! The fiber is not defined
bottom right, top left")

```

```

00766
00767     # check for convexity (vector products)
00768     outIJxIK = (Jy-Iy)*(Kz-Iz) - (Ky-Iy)*(Jz-Iz)
00769     outIKxIL = (Ky-Iy)*(Lz-Iz) - (Ly-Iy)*(Kz-Iz)
00770     # check if I, J, L points are colinear
00771     outIJxIL = (Jy-Iy)*(Lz-Iz) - (Ly-Iy)*(Jz-Iz)
00772     # outJKxJL = (Ky-Jy)*(Lz-Jz) - (Ly-Jy)*(Kz-Jz)
00773
00774     if -outIJxIK <= 0 or -outIKxIL <= 0 or -outIJxIL <= 0:
00775         print('!!!!!!! WARNING !!!!!!! Patch quad is non-convex or non-counter-clockwise
defined or has at least 3 colinear points in line')
00776
00777     IJz, IJy = np.linspace(Iz, Jz, nIJ+1), np.linspace(Iy, Jy, nIJ+1)
00778     JKz, JKy = np.linspace(Jz, Kz, nJK+1), np.linspace(Jy, Ky, nJK+1)
00779     LKz, LKy = np.linspace(Lz, Kz, nIJ+1), np.linspace(Ly, Ky, nIJ+1)
00780     ILz, ILy = np.linspace(Iz, Lz, nJK+1), np.linspace(Iy, Ly, nJK+1)
00781
00782     if fill_shapes:
00783         Z = np.zeros((nIJ+1, nJK+1))
00784         Y = np.zeros((nIJ+1, nJK+1))
00785
00786         for j in range(nIJ+1):
00787             Z[j, :] = np.linspace(IJz[j], LKz[j], nJK+1)
00788             Y[j, :] = np.linspace(IJy[j], LKy[j], nJK+1)
00789
00790         for j in range(nIJ):
00791             for k in range(nJK):
00792                 zy = np.array([[ -Z[j, k], Y[j, k]],
00793                               [ -Z[j, k+1], Y[j, k+1]],
00794                               [ -Z[j+1, k+1], Y[j+1, k+1]],
00795                               [ -Z[j+1, k], Y[j+1, k]]])
00796                 poly = Polygon(zy, True, ec='k', fc=mat_to_col[matID])
00797                 ax.add_patch(poly)
00798
00799     else:
00800         # horizontal lines
00801         for az, bz, ay, by in zip(IJz, LKz, IJy, LKy):
00802             plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)
00803
00804         # vertical lines
00805         for az, bz, ay, by in zip(JKz, ILz, JKy, ILy):
00806             plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)
00807
00808     if item[0] == 'patch' and item[1] == 'circ':
00809         matID, nc, nr = item[2], item[3], item[4]
00810         mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
00811
00812         yC, zC, ri, re = item[5], item[6], item[7], item[8]
00813         if len(item) > 9:
00814             a0 = item[9]
00815         else:
00816             a0 = 0.
00817         a1 = 360. + a0
00818         if len(item) > 10: a1 = item[10]
00819
00820         dr = (re - ri) / nr
00821         dth = (a1 - a0) / nc
00822
00823         for j in range(nr):
00824             rj = ri + j * dr
00825             rj1 = rj + dr
00826
00827             for i in range(nc):
00828                 thi = a0 + i * dth
00829                 th1 = thi + dth
00830                 wedge = Wedge((-zC, yC), rj1, thi, th1, width=dr, ec='k', #Seweryn Kokot: (yC,
-zC), wrong??
00831                               lw=1, fc=mat_to_col[matID])
00832                 ax.add_patch(wedge)
00833                 ax.set(xlabel='x dimension {}'.format(length_unit), ylabel='y dimension
[{}].format(length_unit),
00834                       title='Fiber section (ID = {})'.format(fib_ID))
00835                 ax.axis('equal')
00836
00837

```

## 6.7 FunctionalFeatures Namespace Reference

Module with useful functions (discretise curves, ID conventions, etc)  
Carmin Schipani, 2021.



## Classes

- class [IDGenerator](#)  
*Class that manage the ID generation.*

## Functions

- def [DiscretizeLinearly](#) (np.ndarray LP, int discr, plot=False, block=False, show\_original\_LP=True)  
*This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).*
- def [DiscretizeLoadProtocol](#) (np.ndarray SDR\_LP, np.ndarray nr\_cycles\_LP, int discr\_first\_cycle, plot=False, block=False, show\_original\_peaks=True)  
*Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).*
- def [GridIDConvention](#) (int pier\_axis, int floor\_axis, max\_pier=-1, max\_floor=-1)  
*Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).*
- def [IDConvention](#) (int prefix, int suffix, n\_zeros\_between=0)  
*Function used to construct IDs for elements and offgrid nodes.*
- def [NodesOrientation](#) (int iNode\_ID, int jNode\_ID)  
*Function that finds the orientation of the vector with direction 'jNode\_ID'-'iNode\_ID'.*
- def [OffsetNodeIDConvention](#) (int node\_ID, str orientation, str position\_i\_or\_j)  
*Function used to add node on top of existing ones in the extremes of members with springs.*
- def [plot\\_member](#) (list element\_array, member\_name="Member name not defined", show\_element\_ID=True, show\_node\_ID=True)  
*Function that plots a set of elements.*
- def [plot\\_nodes](#) (list nodes\_array, name="Not defined", show\_node\_ID=True)  
*Function that plots a set of nodes.*
- def [ProgressingPercentage](#) (max\_iter, int i, int next\_step, step=10)  
*Function that shows the progressing percentage of an iterative process.*

### 6.7.1 Detailed Description

Module with useful functions (discretise curves, ID conventions, etc)  
Carmin Schipani, 2021.

### 6.7.2 Function Documentation

#### 6.7.2.1 DiscretizeLinearly()

```
def FunctionalFeatures.DiscretizeLinearly (
    np.ndarray LP,
    int discr,
    plot = False,
    block = False,
    show_original_LP = True )
```

This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).

## Parameters

<i>LP</i>	(np.ndarray): Array (1 dimension) that stores the curve that needs to be discretized
<i>discr</i>	(int): The number of points to add between every two points of 'LP' (linearly)
<i>plot</i>	(bool, optional): Option to show the plot of the discretized (and also the original LP). Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
<i>show_original_LP</i>	(bool, optional): Option to show the original LP to check if the discretized curve is correct. Defaults to True.

## Returns

np.ndarray: Array (1 dimension) that stores the new discretized load protocol.

Definition at line 99 of file [FunctionalFeatures.py](#).

```

00099 def DiscretizeLinearly(LP: np.ndarray, discr: int, plot = False, block = False, show_original_LP =
    True):
00100     """
00101     This function discretize the curve 'LP' given adding the number of point given by 'discr' between
    every point (linearly).
00102
00103     @param LP (np.ndarray): Array (1 dimension) that stores the curve that needs to be discretized
00104     @param discr (int): The number of points to add between every two points of 'LP' (linearly)
00105     @param plot (bool, optional): Option to show the plot of the discretized (and also the original
    LP). Defaults to False.
00106     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of
    the program everytime that a plot should pop up). Defaults to False.
00107     @param show_original_LP (bool, optional): Option to show the original LP to check if the
    discretized curve is correct. Defaults to True.
00108
00109     @returns np.ndarray: Array (1 dimension) that stores the new discretized load protocol.
00110     """
00111
00112     #TODO: check discr nonnegative int and LP 1 dimension
00113
00114     # Define the new discretized LP
00115     length = 1 + (np.size(LP)-1) * (discr+1)
00116     discr_LP = np.zeros(length)
00117
00118     # Performa manually the first iteration
00119     yprev = LP[0]
00120     x = [0, 1]
00121     discr_LP[0] = yprev
00122     iter = 0
00123
00124     # add the other points and the discretized ones
00125     for ynext in LP[1:]:
00126         y = [yprev, ynext]
00127
00128         # Compute new points
00129         xnew = np.linspace(x[0], x[1], discr+2)
00130         ynew = np.interp(xnew[1:], x, y)
00131
00132         # Add to the recording vector discr_LP
00133         index = np.array(np.arange(discr+1)+1+iter)
00134         discr_LP[index] = ynew
00135
00136         # Prepare for next iteration
00137         yprev = ynext
00138         iter = iter + discr + 1
00139
00140     if plot:
00141         fig, ax = plt.subplots()
00142         ax.plot(discr_LP, '-x', label='Discretised LP')
00143
00144         ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
00145               title='Discretized loading protocol')
00146         ax.grid()
00147
00148     if show_original_LP:
00149         x_val = np.arange(0, np.size(discr_LP), discr+1)
00150         ax.plot(x_val, LP, 'ob', label='Original LP')
00151         ax.legend()
00152
00153     if block:

```

```

00154         plt.show()
00155
00156     return discr_LP
00157
00158

```

### 6.7.2.2 DiscretizeLoadProtocol()

```

def FunctionalFeatures.DiscretizeLoadProtocol (
    np.ndarray SDR_LP,
    np.ndarray nr_cycles_LP,
    int discr_first_cycle,
    plot = False,
    block = False,
    show_original_peaks = True )

```

Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).

#### Parameters

<i>SDR_LP</i>	(np.ndarray): Array (1 dimension) that stores the peaks of the cycles. They needs to be only the positive peaks, beacuse this function will use them as the extreme for each cycle.
<i>nr_cycles_LP</i>	(np.ndarray): Array (1 dimension) that stores the number of cycles for every extreme declared in 'SDR_LP' and its countepart negative. They need to be positive integers.
<i>discr_first_cycle</i>	(int): The number of points from peak to peak (counting the two peaks) in the first cycle. It should be odd.
<i>plot</i>	(bool, optional): Option to show the plot of the discretized (and also the original peaks). Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
<i>show_original_peaks</i>	(bool, optional): Option to show the original peaks to check if the discretized curve is correct. The argument plot need to be True. Defaults to True.

#### Exceptions

<i>WrongDimension</i>	SDR_LP and nr_cycles_LP need to be of same length.
<i>NegativeValue</i>	SDR_LP needs to have only positive integers.
<i>NegativeValue</i>	nr_cycles_LP needs to have only positive integers.
<i>NegativeValue</i>	discr_first_cycle needs to be a positive integer.

#### Returns

np.array: Array (1 dimension) that stores the new discretized load protocol curve.

Definition at line 32 of file [FunctionalFeatures.py](#).

```

00032 def DiscretizeLoadProtocol(SDR_LP: np.ndarray, nr_cycles_LP: np.ndarray, discr_first_cycle: int, plot
    = False, block = False, show_original_peaks = True):
00033     """
00034     Discretized a cyclic load protocol keeping a similar discretisation step throughout the different
    cycles and keeping in the output the extremes (peaks).
00035

```

```

00036     @param SDR_LP (np.ndarray): Array (1 dimension) that stores the peaks of the cycles.
00037     They needs to be only the positive peaks, beacuse this function will use them as the extreme
00038     for each cycle.
00038     @param nr_cycles_LP (np.ndarray): Array (1 dimension) that stores the number of cycles for every
00039     extreme declared in 'SDR_LP' and its countepart negative.
00039     They need to be positive integers.
00040     @param discr_first_cycle (int): The number of points from peak to peak (counting the two peaks) in
00041     the first cycle. It should be odd.
00041     @param plot (bool, optional): Option to show the plot of the discretized (and also the original
00042     peaks). Defaults to False.
00042     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of
00043     the program everytime that a plot should pop up). Defaults to False.
00043     @param show_original_peaks (bool, optional): Option to show the original peaks to check if the
00044     discretized curve is correct.
00044     The argument plot need to be True. Defaults to True.
00045
00046     @exception WrongDimension: SDR_LP and nr_cycles_LP need to be of same length.
00047     @exception NegativeValue: SDR_LP needs to have only positive integers.
00048     @exception NegativeValue: nr_cycles_LP needs to have only positive integers.
00049     @exception NegativeValue: discr_first_cycle needs to be a positive integer.
00050
00051     @returns np.array: Array (1 dimension) that stores the new discretized load protocol curve.
00052     """
00053     if np.size(SDR_LP) != np.size(nr_cycles_LP): raise WrongDimension()
00054     if any(col < 0 for col in SDR_LP): raise NegativeValue()
00055     if any(col < 0 for col in nr_cycles_LP): raise NegativeValue()
00056     if discr_first_cycle < 0: raise NegativeValue()
00057
00058     if discr_first_cycle % 2 == 0:
00059         discr_first_cycle = discr_first_cycle + 1
00060     discr_factor = discr_first_cycle / (SDR_LP[0]*2)
00061     discretized_LP = [0.0]
00062     x_val = []
00063     skip_x = 0
00064     for i in range(np.size(SDR_LP)):
00065         discr_i = math.ceil(discr_factor*SDR_LP[i]*2)-1;
00066         if discr_i % 2 == 0:
00067             discr_i = discr_i + 1
00068         length_tmp = int((discr_i+1)/2)
00069         tmp_up = np.linspace(0.0, SDR_LP[i], length_tmp)
00070         tmp_down = np.linspace(SDR_LP[i], 0.0, length_tmp)
00071         for j in range(int(nr_cycles_LP[i])):
00072             discretized_LP = np.append(discretized_LP, tmp_up[1:length_tmp])
00073             discretized_LP = np.append(discretized_LP, tmp_down[1:length_tmp])
00074             discretized_LP = np.append(discretized_LP, -tmp_up[1:length_tmp])
00075             discretized_LP = np.append(discretized_LP, -tmp_down[1:length_tmp])
00076         # for check of original peaks
00077         x_val.append(length_tmp-1+skip_x)
00078         skip_x = (length_tmp-1)*(4*(nr_cycles_LP[i]-1)+3)+x_val[-1]
00079
00080
00081     if plot:
00082         fig, ax = plt.subplots()
00083         ax.plot(discretized_LP, '-r', label='Discretised LP')
00084
00085         ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
00086               title='Discretized loading protocol')
00087         ax.grid()
00088
00089     if show_original_peaks:
00090         ax.plot(x_val, SDR_LP, 'ob', label='Original LP')
00091         ax.legend()
00092
00093     if block:
00094         plt.show()
00095
00096     return discretized_LP
00097
00098

```

### 6.7.2.3 GridIDConvention()

```

def FunctionalFeatures.GridIDConvention (
    int pier_axis,
    int floor_axis,
    max_pier = -1,
    max_floor = -1 )

```

Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).

The conventional grid node ID is xy, with x = the pier position 'pier\_axis'; y = the floor position 'floor\_axis'.

#### Parameters

<i>pier_axis</i>	(int): The pier (or x) postion of the node.
<i>floor_axis</i>	(int): The floor (or y) position of the node.
<i>max_pier</i>	(int, optional): Maximal pier position of the model (used to identify the number of digits). Defaults to -1, e.g. taken equal of pier_axis.
<i>max_floor</i>	(int, optional): Maximal floor position of the model (used to identify the number of digits). Defaults to -1, e.g. taken equal of floor_axis.

#### Exceptions

<i>NameError</i>	Work In Progress: only 9 floors/bays.
<i>NegativeValue</i>	The argument pier_axis needs to be a positive integer.
<i>NegativeValue</i>	The argument floor_axis needs to be a positive integer.
<i>NegativeValue</i>	The argument max_pier needs to be a positive integer if different from -1.
<i>NegativeValue</i>	The argument max_floor needs to be a positive integer if different from -1.
<i>WrongArgument</i>	The argument max_pier need to be equal or bigger to pier_axis
<i>WrongArgument</i>	The argument max_floor need to be equal or bigger to floor_axis

#### Returns

int: The grid node ID

Definition at line 159 of file [FunctionalFeatures.py](#).

```

00159 def GridIDConvention(pier_axis: int, floor_axis: int, max_pier = -1, max_floor = -1):
00160     """
00161     Function used to construct the ID of the nodes in the grid (first nodes that define the geometry
    of the model).
00162     The conventional grid node ID is xy, with x = the pier position 'pier_axis'; y = the floor
    position 'floor_axis'.
00163
00164     @param pier_axis (int): The pier (or x) postion of the node.
00165     @param floor_axis (int): The floor (or y) position of the node.
00166     @param max_pier (int, optional): Maximal pier position of the model (used to identify the number
    of digits).
00167     Defaults to -1, e.g. taken equal of pier_axis.
00168     @param max_floor (int, optional): Maximal floor position of the model (used to identify the number
    of digits).
00169     Defaults to -1, e.g. taken equal of floor_axis.
00170
00171     @exception NameError: Work In Progress: only 9 floors/bays.
00172     @exception NegativeValue: The argument pier_axis needs to be a positive integer.
00173     @exception NegativeValue: The argument floor_axis needs to be a positive integer.
00174     @exception NegativeValue: The argument max_pier needs to be a positive integer if different from
    -1.
00175     @exception NegativeValue: The argument max_floor needs to be a positive integer if different from
    -1.
00176     @exception WrongArgument: The argument max_pier need to be equal or bigger to pier_axis
00177     @exception WrongArgument: The argument max_floor need to be equal or bigger to floor_axis
00178
00179     @returns int: The grid node ID
00180     """
00181     # Convention:
00182     #     GridNodeID:    xy          with x = pier, y = floor
00183     if pier_axis > 9 or floor_axis > 9 or max_pier > 9 or max_floor > 9: raise NameError("WIP: maximal
    9 floors or bays")
00184     max_pier = pier_axis if max_pier == -1 else max_pier
00185     max_floor = floor_axis if max_floor == -1 else max_floor
00186
00187     if pier_axis < 0: raise NegativeValue()
00188     if floor_axis < 0: raise NegativeValue()
00189     if max_pier < 0: raise NegativeValue()

```

```

00190         if max_floor < 0: raise NegativeValue()
00191         if max_pier < pier_axis: raise WrongArgument()
00192         if max_floor < floor_axis: raise WrongArgument()
00193
00194         max_x_digits = int(math.log10(max_pier))+1
00195         max_y_digits = int(math.log10(max_floor))+1
00196
00197         # return 10**(max_x_digits+max_y_digits) + pier_axis*10**max_y_digits + floor_axis # with 1 as
00198         # prefix (to consider more than one digit per axis, but exceed max ID)
00199         return pier_axis*10**max_y_digits + floor_axis
00200

```

#### 6.7.2.4 IDConvention()

```

def FunctionalFeatures.IDConvention (
    int prefix,
    int suffix,
    n_zeros_between = 0 )

```

Function used to construct IDs for elements and offgrid nodes.

It appends to a positive integer number 'prefix' a number of zeros 'n\_zeros\_between' and at last another positive integer 'suffix'. The conventional element ID is xy(a)x'y'(a') with xya = the node ID in pier x, floor y and offgrid parameter a (optional); x'y'a' = the node ID in pier x', floor y' and offgrid parameter a' (optional). For more information on x and y, see GridIDConvention; for more information on a, see OffsetNodeIDConvention.

##### Parameters

<i>prefix</i>	(int): Prefix of the new ID. For a vertical element it should be the left node ID; for an horizontal one it should be the bottom node.
<i>suffix</i>	(int): Suffix of the new ID. For a vertical element it should be the right node ID; for an horizontal one it should be the top node.
<i>n_zeros_between</i>	(int, optional): Number of zeros to add between the two nodes. Defaults to 0.

##### Exceptions

<i>NegativeValue</i>	The argument prefix needs to be a positive integer.
<i>NegativeValue</i>	The argument suffix needs to be a positive integer.
<i>NegativeValue</i>	The argument n_zeros_between needs to be a positive integer.

##### Returns

int: The combined ID

Definition at line 201 of file [FunctionalFeatures.py](#).

```

00201 def IDConvention(prefix: int, suffix: int, n_zeros_between = 0):
00202     """
00203     Function used to construct IDs for elements and offgrid nodes.
00204     It appends to a positive integer number 'prefix' a number of zeros 'n_zeros_between' and at last
00205     another positive integer 'suffix'.
00206     The conventional element ID is xy(a)x'y'(a') with xya = the node ID in pier x, floor y and offgrid
00207     parameter a (optional);
00208     x'y'a' = the node ID in pier x', floor y' and offgrid parameter a' (optional).
00209     For more information on x and y, see GridIDConvention; for more information on a, see
00210     OffsetNodeIDConvention.
00211
00212     @param prefix (int): Prefix of the new ID. For a vertical element it should be the left node ID;

```

```

00210         for an horizontal one it should be the bottom node.
00211     @param suffix (int): Suffix of the new ID. For a vertical element it should be the right node ID;
00212         for an horizontal one it should be the top node.
00213     @param n_zeros_between (int, optional): Number of zeros to add between the two nodes. Defaults to
00214         0.
00215     @exception NegativeValue: The argument prefix needs to be a positive integer.
00216     @exception NegativeValue: The argument suffix needs to be a positive integer.
00217     @exception NegativeValue: The argument n_zeros_between needs to be a positive integer.
00218
00219     @returns int: The combined ID
00220     """
00221     # Convention:
00222     #     ElementID:      xy(a)x'y' (a')    with xy(a) = NodeID i and x'y' (a') = NodeID j
00223     #     TrussID:        xy(a)x'y' (a')    with xy(a) = NodeID i and x'y' (a') = NodeID j
00224     #     Spring:         xy(a)x'y' (a')    with xy(a) = NodeID i and x'y' (a') = NodeID j
00225     if prefix < 0: raise NegativeValue()
00226     if suffix < 0: raise NegativeValue()
00227     if n_zeros_between < 0: raise NegativeValue()
00228
00229     return int(str(prefix*10**n_zeros_between) + str(suffix))
00230
00231

```

### 6.7.2.5 NodesOrientation()

```

def FunctionalFeatures.NodesOrientation (
    int iNode_ID,
    int jNode_ID )

```

Function that finds the orientation of the vector with direction 'jNode\_ID"iNode\_ID'.

If the the nodes are on top of each other, the function returns 'zero\_length'.

#### Parameters

<i>iNode_ID</i>	(int): Node i.
<i>jNode_ID</i>	(int): Node j.

#### Exceptions

<i>NegativeValue</i>	The argument iNode_ID needs to be a positive integer.
<i>NegativeValue</i>	The argument jNode_ID needs to be a positive integer.

#### Returns

str: The orientation of the vector.

Definition at line 270 of file [FunctionalFeatures.py](#).

```

00270 def NodesOrientation(iNode_ID: int, jNode_ID: int):
00271     """
00272     Function that finds the orientation of the vector with direction 'jNode_ID"iNode_ID'.
00273     If the the nodes are on top of each other, the function returns 'zero_length'.
00274
00275     @param iNode_ID (int): Node i.
00276     @param jNode_ID (int): Node j.
00277
00278     @exception NegativeValue: The argument iNode_ID needs to be a positive integer.
00279     @exception NegativeValue: The argument jNode_ID needs to be a positive integer.
00280
00281     @returns str: The orientation of the vector.
00282     """

```

```

00283         if iNode_ID < 1: raise NegativeValue()
00284         if jNode_ID < 1: raise NegativeValue()
00285
00286         iNode = np.array(nodeCoord(iNode_ID))
00287         jNode = np.array(nodeCoord(jNode_ID))
00288         if abs(iNode[0]-jNode[0]) + abs(iNode[1]-jNode[1]) == 0:
00289             return "zero_length"
00290         elif abs(iNode[0]-jNode[0]) < abs(iNode[1]-jNode[1]):
00291             return "vertical"
00292         else:
00293             return "horizontal"
00294
00295

```

### 6.7.2.6 OffsetNodeIDConvention()

```

def FunctionalFeatures.OffsetNodeIDConvention (
    int node_ID,
    str orientation,
    str position_i_or_j )

```

Function used to add node on top of existing ones in the extremes of memembers with springs.

#### Parameters

<i>node_ID</i>	(int): Node that we refer to.
<i>orientation</i>	(str): Orientation of the memeber. Can be 'vertical' or 'horizontal'.
<i>position_i_↔ or_j</i>	(str): Position at the start 'i' (left or bottom) or at the end 'j' (right or top) of 'node_ID' in the member.

#### Exceptions

<i>NegativeValue</i>	The argument node_ID needs to be a positive integer.
<i>WrongArgument</i>	The argument position_i_or_j needs to be 'i' or 'j'
<i>WrongArgument</i>	The argument orientation needs to be 'vertical' or 'horizontal'

#### Returns

int: The combined ID

Definition at line 232 of file [FunctionalFeatures.py](#).

```

00232 def OffsetNodeIDConvention(node_ID: int, orientation: str, position_i_or_j: str):
00233     """
00234     Function used to add node on top of existing ones in the extremes of memembers with springs.
00235
00236     @param node_ID (int): Node that we refer to.
00237     @param orientation (str): Orientation of the memeber. Can be 'vertical' or 'horizontal'.
00238     @param position_i_or_j (str): Position at the start 'i' (left or bottom)
00239         or at the end 'j' (right or top) of 'node_ID' in the member.
00240
00241     @exception NegativeValue: The argument node_ID needs to be a positive integer.
00242     @exception WrongArgument: The argument position_i_or_j needs to be 'i' or 'j'
00243     @exception WrongArgument: The argument orientation needs to be 'vertical' or 'horizontal'
00244
00245     @returns int: The combined ID
00246     """
00247     # Convention:
00248     #   o xy          GridNodeID:      xy          with x = pier, y = floor
00249     #   o xy7

```



```

00249      # AdditionalNodeID:      xya          with x = pier, y = floor, a:  o xy  xy2 o-----o x'y3
      x'y o |
00250      # PanelZoneNodeID:      xy(a)a      see MemberModel for the panel zone ID convention
      |
00251      #
      |
00252      #
      o xy'6
00253      #
      o xy'
00254      if node_ID < 1: raise NegativeValue()
00255      if position_i_or_j != "i" and position_i_or_j != "j": raise WrongArgument()
00256
00257      if orientation == "vertical":
00258          if position_i_or_j == "i":
00259              return IDConvention(node_ID, 6)
00260          else:
00261              return IDConvention(node_ID, 7)
00262      elif orientation == "horizontal":
00263          if position_i_or_j == "i":
00264              return IDConvention(node_ID, 2)
00265          else:
00266              return IDConvention(node_ID, 3)
00267      else: raise WrongArgument()
00268
00269

```

### 6.7.2.7 plot\_member()

```

def FunctionalFeatures.plot_member (
    list element_array,
    member_name = "Member name not defined",
    show_element_ID = True,
    show_node_ID = True )

```

Function that plots a set of elements.

It can be used to check the correctness of a part of the model or of a member. If the entire model need to be plotted, use instead 'plot\_model("nodes", "elements")' from `openseespy.postprocessing.Get_Rendering`. Inspired by `plot_model` written by Anurag Upadhyay and Christian Slotboom.

#### Parameters

<i>element_array</i>	(list): An array (list of lists of one dimensions and length = 3) that store the element and nodes IDs. An element is stored in one list with 3 entries: the element ID, node i ID and node j ID.
<i>member_name</i>	(str, optional): The name of what is plotted. Defaults to "Member name not defined".
<i>show_element_ID</i>	(bool, optional): Option to show the element IDs. Defaults to True.
<i>show_node_ID</i>	(bool, optional): Option to show the nodes IDs. Defaults to True.

#### Exceptions

<i>WrongDimension</i>	<code>element_array</code> needs to be non-empty.
<i>WrongDimension</i>	The number of entries in the lists inside the argument <code>element_array</code> need to be 3.

#### Returns

`matplotlib.axes._subplots.AxesSubplo`: The figure's wrapper, useful to customise the plot (change axes label, etc).

Definition at line 296 of file [FunctionalFeatures.py](#).

```

00296 def plot_member(element_array: list, member_name = "Member name not defined", show_element_ID = True,
00297                  show_node_ID = True):
00298     """
00299     Function that plots a set of elements. It can be used to check the correctness of a part of the
00300     model or of a member.
00301     If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from
00302     openseespy.postprocessing.Get_Rendering. \n
00303     Inspired by plot_model written by Anurag Upadhyay and Christian Slotboom.
00304
00305     @param element_array (list): An array (list of lists of one dimensions and length = 3) that store
00306     the element and nodes IDs.
00307     An element is stored in one list with 3 entries: the element ID, node i ID and node j ID.
00308     @param member_name (str, optional): The name of what is plotted. Defaults to "Member name not
00309     defined".
00310     @param show_element_ID (bool, optional): Option to show the element IDs. Defaults to True.
00311     @param show_node_ID (bool, optional): Option to show the nodes IDs. Defaults to True.
00312
00313     @exception WrongDimension: element_array needs to be non-empty.
00314     @exception WrongDimension: The number of entries in the lists inside the argument element_array
00315     need to be 3.
00316
00317     @returns matplotlib.axes._subplots.AxesSubplo: The figure's wrapper, useful to customise the plot
00318     (change axes label, etc).
00319     """
00320     if len(element_array) == 0: raise WrongArgument()
00321     if len(element_array[0]) != 3: raise WrongDimension()
00322
00323     node_style = {'color':'black', 'marker':'o', 'facecolor':'black', 'linewidth':0.}
00324     node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
00325     track_node = {}
00326
00327     if show_element_ID:
00328         show_e_ID = 'yes'
00329     else:
00330         show_e_ID = 'no'
00331
00332     fig = plt.figure()
00333     ax = fig.add_subplot(1,1,1)
00334
00335     for ele in element_array:
00336         eleTag = int(ele[0])
00337         Nodes =ele[1:]
00338
00339         if len(Nodes) == 2:
00340             # 2D element
00341             iNode = np.array(nodeCoord(Nodes[0]))
00342             jNode = np.array(nodeCoord(Nodes[1]))
00343             ipltf._plotBeam2D(iNode, jNode, ax, show_e_ID, eleTag, "solid")
00344             ax.scatter(*iNode, **node_style)
00345             ax.scatter(*jNode, **node_style)
00346             if show_node_ID:
00347                 if abs(sum(iNode - jNode)) > 1e-6:
00348                     # beam-col
00349                     _plt_node(Nodes[0], track_node, iNode, ax, node_text_style)
00350                     _plt_node(Nodes[1], track_node, jNode, ax, node_text_style, h_align='right',
00351                             v_align='bottom')
00352                 else:
00353                     # zerolength
00354                     _plt_node(Nodes[0], track_node, iNode, ax, node_text_style, h_align='right')
00355                     _plt_node(Nodes[1], track_node, jNode, ax, node_text_style)
00356             else:
00357                 print("Too many nodes in this elemnet (see shell elements)")
00358
00359     ax.set_xlabel('x [{}].format(length_unit))
00360     ax.set_ylabel('y [{}].format(length_unit))
00361     plt.title("Visualisation of: {}".format(member_name))
00362     plt.axis('equal')
00363     return ax
00364
00365
00366
00367

```

### 6.7.2.8 plot\_nodes()

```

def FunctionalFeatures.plot_nodes (
    list nodes_array,
    name = "Not defined",
    show_node_ID = True )

```

Function that plots a set of nodes.

It can be used to check the correctness of the model's geometry. If the entire model need to be plotted, use instead 'plot\_model("nodes", "elements")' from `openseespy.postprocessing.Get_Rendering`.

#### Parameters

<i>nodes_array</i>	(list): List of 1 dimension with the IDs of the nodes to be displayed.
<i>name</i>	(str, optional): Name that describe what the plot will show. Defaults to "Not defined".
<i>show_node_ID</i>	(bool, optional): Option to show the node IDs. Defaults to True.

#### Exceptions

<i>WrongArgument</i>	<code>nodes_array</code> needs to be non-empty.
----------------------	---

#### Returns

(`matplotlib.axes._subplots.AxesSubplot`): The figure's wrapper, useful to customise the plot (change axes label, etc).

Definition at line 358 of file [FunctionalFeatures.py](#).

```

00358 def plot_nodes(nodes_array: list, name = "Not defined", show_node_ID = True):
00359     """
00360     Function that plots a set of nodes. It can be used to check the correctness of the model's
00361     geometry.
00362     If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from
00363     openseespy.postprocessing.Get_Rendering.
00364     @param nodes_array (list): List of 1 dimension with the IDs of the nodes to be displayed.
00365     @param name (str, optional): Name that describe what the plot will show. Defaults to "Not
00366     defined".
00367     @param show_node_ID (bool, optional): Option to show the node IDs. Defaults to True.
00368     @exception WrongArgument: nodes_array needs to be non-empty.
00369     @returns (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper, useful to customise the
00370     plot (change axes label, etc).
00371     """
00372     if len(nodes_array) == 0: raise WrongArgument()
00373     node_style = {'color':'black', 'marker':'o', 'facecolor':'black', 'linewidth':0.}
00374     node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
00375     track_node = {}
00376
00377     fig = plt.figure()
00378     ax = fig.add_subplot(1,1,1)
00379
00380     for node_ID in nodes_array:
00381         node_xy = np.array(nodeCoord(node_ID))
00382         ax.scatter(*node_xy, **node_style)
00383         if show_node_ID:
00384             __plt_node(node_ID, track_node, node_xy, ax, node_text_style)
00385
00386     ax.set_xlabel('x [{}].format(length_unit))
00387     ax.set_ylabel('y [{}].format(length_unit))
00388     plt.title("Visualisation of: {}".format(name))
00389     plt.axis('equal')
00390     return ax
00391
00392

```

#### 6.7.2.9 ProgressingPercentage()

```

def FunctionalFeatures.ProgressingPercentage (
    max_iter,

```

```

    int i,
    int next_step,
    step = 10 )

```

Function that shows the progressing percentage of an iterative process.

#### Parameters

<i>max_iter</i>	(int): Maximal number of iterations
<i>i</i>	(int): Current iteration
<i>next_step</i>	(int): Next step of the percentage (set to 0 for the first iteration and then use the return parameter)
<i>step</i>	(int, optional): Size of the step (should be a fraction of 100). Defaults to 10.

#### Returns

int: The updated next step

Definition at line 14 of file [FunctionalFeatures.py](#).

```

00014 def ProgressingPercentage(max_iter, i: int, next_step: int, step = 10):
00015     """
00016     Function that shows the progressing percentage of an iterative process.
00017
00018     @param max_iter (int): Maximal number of iterations
00019     @param i (int): Current iteration
00020     @param next_step (int): Next step of the percentage (set to 0 for the first iteration and then use
the return parameter)
00021     @param step (int, optional): Size of the step (should be a fraction of 100). Defaults to 10.
00022
00023     @returns int: The updated next step
00024     """
00025     if i*100.0/(max_iter-1) >= next_step:
00026         print("The progression is {}".format(next_step))
00027         return next_step + step
00028
00029     return next_step
00030
00031

```

## 6.8 GeometryTemplate Namespace Reference

Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).

### Functions

- def [DefineFrameNodes](#) (int n\_hor\_axis, int n\_vert\_axis, storey\_width, storey\_height, half\_pz\_height=np.array([ ]), origin=[0, 0], first\_hor\_axis=1, first\_vert\_axis=1, show\_plot=True)  
*Function that declares and initialises the grid nodes of a frame.*
- def [DefineFrameNodesAndElementsSteellShape](#) (int n\_hor\_axis, int n\_vert\_axis, storey\_width, storey\_height, list list\_col, list list\_beam, int geo\_trans\_ID, N\_G=np.array([ ]), t\_dp=np.array([ ]), L\_b\_col=np.array([ ]), L\_b\_beam=np.array([ ]), fix\_support=True, show\_plot=True, panel\_zone=True)  
*WIP (Work In Progress).*
- def [DefineSubassemblageNodes](#) (beam\_left\_L\_cl, beam\_right\_L\_cl, col\_top\_L\_cl, col\_bottom\_L\_cl, depth\_col, depth\_beam, boundary\_condition=True, show\_plot=True)  
*Function that declares and initialises the grid nodes of an interior subassemblage.*
- def [Initialize2DModel](#) (data\_dir="Results")  
*Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.*

## 6.8.1 Detailed Description

Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).

Carmine Schipani, 2021

## 6.8.2 Function Documentation

### 6.8.2.1 DefineFrameNodes()

```
def GeometryTemplate.DefineFrameNodes (
    int n_hor_axis,
    int n_vert_axis,
    storey_width,
    storey_height,
    half_pz_height = np.array([]),
    origin = [0, 0],
    first_hor_axis = 1,
    first_vert_axis = 1,
    show_plot = True )
```

Function that declares and initialises the grid nodes of a frame.

Option to offset the grid node of the panel zones with the master node of the panel zone being the grid one (top center one). The function can be used multiple times to create more complex geometries.

#### Parameters

<i>n_hor_axis</i>	(int): Number of horizontal axis (or piers) for the grid of the frame.
<i>n_vert_axis</i>	(int): Number of vertical axis (or floors) for the grid of the frame.
<i>storey_width</i>	(float): Width of the bays.
<i>storey_height</i>	(float): Height of the storeys.
<i>half_pz_height</i>	(np.ndarray, optional): Array of 1 dimension with half the height of the panel zone for each floor. The first floor should be 0 (no panel zone in the support). Defaults to np.array([]), e.g. no panel zone.
<i>origin</i>	(list, optional): List of two entry with the origin position. Defaults to [0, 0].
<i>first_hor_axis</i>	(int, optional): Number of the first pier. Defaults to 1.
<i>first_vert_axis</i>	(int, optional): Number of the first floor. Defaults to 1.
<i>show_plot</i>	(bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

#### Exceptions

<i>NegativeValue</i>	<i>n_hor_axis</i> needs to be a positive integer.
<i>NegativeValue</i>	<i>n_vert_axis</i> needs to be a positive integer.
<i>NegativeValue</i>	<i>storey_width</i> needs to be positive.
<i>NegativeValue</i>	<i>storey_height</i> needs to be positive.
<i>WrongDimension</i>	<i>origin</i> has a dimension of 2.
<i>NegativeValue</i>	<i>first_hor_axis</i> needs to be a positive integer.

## Exceptions

<i>NegativeValue</i>	first_vert_axis needs to be a positive integer.
<i>WrongDimension</i>	size of half_pz_height needs to be equal to n_vert_axis, if different from 0.

## Returns

list: List with the nodes declared.

Definition at line 44 of file [GeometryTemplate.py](#).

```

00045     origin = [0, 0], first_hor_axis = 1, first_vert_axis = 1, show_plot = True):
00046     """
00047     Function that declares and initialises the grid nodes of a frame. Option to offset the grid node
of the panel zones
00048     with the master node of the panel zone being the grid one (top center one). The function can
be used multiple times
00049     to create more complex geometries.
00050
00051     @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
00052     @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
00053     @param storey_width (float): Width of the bays.
00054     @param storey_height (float): Height of the storeys.
00055     @param half_pz_height (np.ndarray, optional): Array of 1 dimension with half the height of the
panel zone for each floor.
00056     The first floor should be 0 (no panel zone in the support). Defaults to np.array([]), e.g. no
panel zone.
00057     @param origin (list, optional): List of two entry with the origin position. Defaults to [0, 0].
00058     @param first_hor_axis (int, optional): Number of the first pier. Defaults to 1.
00059     @param first_vert_axis (int, optional): Number of the first floor. Defaults to 1.
00060     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
Defaults to True.
00061
00062     @exception NegativeValue: n_hor_axis needs to be a positive integer.
00063     @exception NegativeValue: n_vert_axis needs to be a positive integer.
00064     @exception NegativeValue: storey_width needs to be positive.
00065     @exception NegativeValue: storey_height needs to be positive.
00066     @exception WrongDimension: origin has a dimension of 2.
00067     @exception NegativeValue: first_hor_axis needs to be a positive integer.
00068     @exception NegativeValue: first_vert_axis needs to be a positive integer.
00069     @exception WrongDimension: size of half_pz_height needs to be equal to n_vert_axis, if different
from 0.
00070
00071     @returns list: List with the nodes declared.
00072     """
00073     if n_hor_axis < 1: raise NegativeValue()
00074     if n_vert_axis < 1: raise NegativeValue()
00075     if storey_width < 0: raise NegativeValue()
00076     if storey_height < 0: raise NegativeValue()
00077     if len(origin) != 2: raise WrongDimension()
00078     if first_hor_axis < 1: raise NegativeValue()
00079     if first_vert_axis < 1: raise NegativeValue()
00080     if np.size(half_pz_height) != 0 and np.size(half_pz_height) != n_vert_axis: raise WrongDimension()
00081
00082     if np.size(half_pz_height) == 0: half_pz_height = np.zeros(n_vert_axis)
00083     node_array = []
00084     max_n_x = n_hor_axis + first_hor_axis - 1
00085     max_n_y = n_vert_axis + first_vert_axis - 1
00086     for xx in range(n_hor_axis):
00087         x_axis = xx + first_hor_axis
00088         for yy in range(n_vert_axis):
00089             y_axis = yy + first_vert_axis
00090             node_ID = GridIDConvention(x_axis, y_axis, max_n_x, max_n_y)
00091             node(node_ID, origin[0]+xx*storey_width, origin[1]+yy*storey_height+half_pz_height[yy])
00092             if y_axis == 1 and half_pz_height[yy] != 0: print("Warning: the nodes at the base have a
panel zone height")
00093             node_array.append(node_ID)
00094
00095     if show_plot:
00096         plot_nodes(node_array, "Frame geometry template with only nodes", True)
00097         plt.grid()
00098
00099     return node_array
00100
00101

```

## 6.8.2.2 DefineFrameNodesAndElementsSteelShape()

```
def GeometryTemplate.DefineFrameNodesAndElementsSteelShape (
    int n_hor_axis,
    int n_vert_axis,
    storey_width,
    storey_height,
    list list_col,
    list list_beam,
    int geo_trans_ID,
    N_G = np.array([]),
    t_dp = np.array([]),
    L_b_col = np.array([]),
    L_b_beam = np.array([]),
    fix_support = True,
    show_plot = True,
    panel_zone = True )
```

WIP (Work In Progress).

Function that declares and initialises the grid nodes of a frame and the members using steel I shape Spring-BasedElements. WARNING: Current limit of the geometry: `n_hor_axis` and `n_vert_axis` < 10; if exceeded, there are problems with the IDs (ID limit is exceeded, ~2.2e9). WARNING: if the section of the columns change, the function does not account for the splicing. Each column section is defined from floor to floor; if there is a change in the column section, it happens right after the panel zone (not realistic but good enough for predesign). WIP: Solve ID limit for large building need implementations (for example the use of a different ID convention or the use of the class IDGenerator).

## Parameters

<code>n_hor_axis</code>	(int): Number of horizontal axis (or piers) for the grid of the frame.
<code>n_vert_axis</code>	(int): Number of vertical axis (or floors) for the grid of the frame.
<code>storey_width</code>	(float): Width of the bays.
<code>storey_height</code>	(float): Height of the storeys.
<code>list_col</code>	(list(SteelShape)): List with the sections of the columns for every floor.
<code>list_beam</code>	(list(SteelShape)): List with the sections of the beams for every bay.
<code>geo_trans_ID</code>	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
<code>N_G</code>	(np.ndarray, optional): Array of dimension 1 with the axial load for each column (starting at floor 2). Defaults to <code>np.array([])</code> , e.g. 0.
<code>t_dp</code>	(np.ndarray, optional): Array of dimension 1 with the doubler plate thickness for each bay's beam. Defaults to <code>np.array([])</code> , e.g. 0.
<code>L_b_col</code>	(np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each column. Defaults to <code>np.array([])</code> , e.g. -1.
<code>L_b_beam</code>	(np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral buckling length for each beam. Defaults to <code>np.array([])</code> , e.g. -1.
<code>fix_support</code>	(bool, optional): Option to fix the support of the frame. Defaults to True.
<code>show_plot</code>	(bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.
<code>panel_zone</code>	(bool, optional): Option to add the panel zones in the model. Defaults to True.

## Exceptions

<code>WrongDimension</code>	<code>N_G</code> dimension needs to be equal to <code>n_vert_axis-1</code> , if different from 0.
<code>WrongDimension</code>	<code>t_dp</code> dimension needs to be equal to <code>n_vert_axis-1</code> , if different from 0.

## Exceptions

<i>WrongDimension</i>	L_b_col dimension needs to be equal to n_vert_axis-1, if different from 0.
<i>WrongDimension</i>	L_b_beam dimension needs to be equal to n_hor_axis-1, if different from 0.
<i>WrongDimension</i>	list_col dimension needs to be equal to n_vert_axis-1.
<i>WrongDimension</i>	list_beam dimension needs to be equal to n_vert_axis-1.
<i>NegativeValue</i>	geo_trans_ID needs to be a positive integer.

## Returns

List: List with the element objects in the frame.

Definition at line 102 of file [GeometryTemplate.py](#).

```

00104     fix_support = True, show_plot = True, panel_zone = True):
00105     """
00106     WIP (Work In Progress). Function that declares and initialises the grid nodes of a frame and the
members using steel I shape SpringBasedElements.
00107     WARNING: Current limit of the geometry: n_hor_axis and n_vert_axis < 10; if exceeded, there are
problems with the IDs (ID limit is exceeded, ~2.2e9).
00108     WARNING: if the section of the columns change, the function does not account for the splacing.
Each colum section is defined from floor to floor;
00109     if there is a change in the column section, it happens right after the panel zone (not realistic
but good enough for predesign).
00110     WIP: Solve ID limit for large building need implementations (for example the use of a different ID
convention or the use of the class IDGenerator).
00111
00112     @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
00113     @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
00114     @param storey_width (float): Width of the bays.
00115     @param storey_height (float): Height of the storeys.
00116     @param list_col (list(SteelIShape)): List with the sections of the columns for every floor.
00117     @param list_beam (list(SteelIShape)): List with the sections of the beams for every bay.
00118     @param geo_trans_ID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
00119     @param N_G (np.ndarray, optional): Array of dimension 1 with the axial load for each column
(starting at floor 2). Defaults to np.array([]), e.g. 0.
00120     @param t_dp (np.ndarray, optional): Array of dimension 1 with the doubler plate thickness for each
bay's beam. Defaults to np.array([]), e.g. 0.
00121     @param L_b_col (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral
buckling length for each column. Defaults to np.array([]), e.g. -1.
00122     @param L_b_beam (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral
buckling length for each beam. Defaults to np.array([]), e.g. -1.
00123     @param fix_support (bool, optional): Option to fix the support of the frame. Defaults to True.
00124     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
Defaults to True.
00125     @param panel_zone (bool, optional): Option to add the panel zones in the model. Defaults to True.
00126
00127     @exception WrongDimension: N_G dimension needs to be equal to n_vert_axis-1, if different from 0.
00128     @exception WrongDimension: t_dp dimension needs to be equal to n_vert_axis-1, if different from 0.
00129     @exception WrongDimension: L_b_col dimension needs to be equal to n_vert_axis-1, if different from
0.
00130     @exception WrongDimension: L_b_beam dimension needs to be equal to n_hor_axis-1, if different from
0.
00131     @exception WrongDimension: list_col dimension needs to be equal to n_vert_axis-1.
00132     @exception WrongDimension: list_beam dimension needs to be equal to n_vert_axis-1.
00133     @exception NegativeValue: geo_trans_ID needs to be a positive integer.
00134
00135     @returns List: List with the element objects in the frame.
00136     """
00137     panel_zone = True
00138     if np.size(N_G) == 0: N_G = np.zeros(n_vert_axis-1)
00139     if np.size(t_dp) == 0: t_dp = np.zeros(n_vert_axis-1)
00140     if np.size(L_b_col) == 0: L_b_col = np.ones(n_vert_axis-1) * (-1.0)
00141     if np.size(L_b_beam) == 0: L_b_beam = np.ones(n_hor_axis-1) * (-1.0)
00142
00143     if np.size(list_col) != n_vert_axis-1: raise WrongDimension()
00144     if np.size(list_beam) != n_vert_axis-1: raise WrongDimension()
00145     if np.size(N_G) != n_vert_axis-1: raise WrongDimension()
00146     if np.size(t_dp) != n_vert_axis-1: raise WrongDimension()
00147     if np.size(L_b_col) != n_vert_axis-1: raise WrongDimension()
00148     if np.size(L_b_beam) != n_hor_axis-1: raise WrongDimension()
00149     if geo_trans_ID < 1: raise NegativeValue()
00150
00151     half_pz_height = np.zeros(n_vert_axis)
00152     if panel_zone:
00153         for ii, beam in enumerate(list_beam):
00154             half_pz_height[ii+1] = beam.d/2

```



```

00155
00156     node_array = DefineFrameNodes(n_hor_axis, n_vert_axis, storey_width, storey_height,
00157                                   half_pz_height, [0, 0], 1, 1, False)
00157
00158     beam_column_pzspring = [[], [], []]
00159     for xx in range(n_hor_axis):
00160         for yy in range(n_vert_axis):
00161             node_ID = node_array[xx*n_vert_axis + yy]
00162             if yy != 0:
00163                 # Panel Zone
00164                 if half_pz_height[yy] == 0:
00165                     col_j_node_ID = node_ID
00166                     beam_j_node_ID = node_ID
00167                 else:
00168                     tmp_pz = PanelZoneSteelIShapeSkiadopoulos2021(node_ID, list_col[yy-1],
00169 list_beam[yy-1], geo_trans_ID, t_dp[yy-1])
00169                     tmp_pz.CreateMember()
00170                     col_j_node_ID = IDConvention(node_ID, 5, 1)
00171                     beam_j_node_ID = IDConvention(node_ID, 8, 1)
00172                     beam_column_pzspring[2].append(deepcopy(tmp_pz))
00173
00174                 # Column
00175                 col_i_node_ID = node_array[xx*n_vert_axis + yy - 1]
00176                 col_mat_i = OffsetNodeIDConvention(col_i_node_ID, "vertical", "i")
00177                 col_mat_j = OffsetNodeIDConvention(col_j_node_ID, "vertical", "j")
00178                 ele_ID = col_i_node_ID if panel_zone else -1
00179                 tmp_col = SpringBasedElementModifiedIMKSteelIShape(col_i_node_ID, col_j_node_ID,
00180 list_col[yy-1], geo_trans_ID,
00181                               col_mat_i, col_mat_j, N_G[yy-1], L_b=L_b_col[yy-1], ele_ID = ele_ID)
00182                 tmp_col.CreateMember()
00183                 beam_column_pzspring[1].append(deepcopy(tmp_col))
00184
00185             if xx != 0:
00186                 # Beam
00187                 if half_pz_height[yy] == 0:
00188                     beam_i_node_ID = node_array[(xx-1)*n_vert_axis + yy]
00189                 else:
00190                     beam_i_node_ID = IDConvention(node_array[(xx-1)*n_vert_axis + yy], 2, 1)
00191                     beam_mat_i = OffsetNodeIDConvention(beam_i_node_ID, "horizontal", "i")
00192                     beam_mat_j = OffsetNodeIDConvention(beam_j_node_ID, "horizontal", "j")
00193                     ele_ID = beam_i_node_ID if panel_zone else -1
00194                     tmp_beam = SpringBasedElementModifiedIMKSteelIShape(beam_i_node_ID,
00195 beam_j_node_ID, list_beam[yy-1], geo_trans_ID,
00196                               beam_mat_i, beam_mat_j, L_b=L_b_beam[xx-1], ele_ID = ele_ID)
00197                     tmp_beam.CreateMember()
00198                     beam_column_pzspring[0].append(deepcopy(tmp_beam))
00199             else:
00200                 if fix_support: RigidSupport(node_ID)
00201
00202         if show_plot:
00203             opsplt.plot_model("nodes", "elements")
00204
00205     return beam_column_pzspring
00206

```

### 6.8.2.3 DefineSubassemblageNodes()

```

def GeometryTemplate.DefineSubassemblageNodes (
    beam_left_L_cl,
    beam_right_L_cl,
    col_top_L_cl,
    col_bottom_L_cl,
    depth_col,
    depth_beam,
    boundary_condition = True,
    show_plot = True )

```

Function that declares and initialises the grid nodes of an interior subassemblage.

The panel zone geometry is defined by the two arguments `depth_col` and `depth_beam`.

## Parameters

<i>beam_left_L_cl</i>	(float): Centerline length of the left beam (excluding the panel zone).
<i>beam_right_L_cl</i>	(float): Centerline length of the right beam (excluding the panle zone).
<i>col_top_L_cl</i>	(float): Centerline length of the top column (excluding the panel zone).
<i>col_bottom_L_cl</i>	(float): Centerline length of the bottom column (excluding the panel zone).
<i>depth_col</i>	(float): Depth of the columns for the panel zone.
<i>depth_beam</i>	(float): Depth of the beams for the panel zone.
<i>boundary_condition</i>	(bool, optional): Option to set already the boundary condition (bottom column pinned, beams fix only y movement). Defaults to True.
<i>show_plot</i>	(bool, optional): Option to show the plot of the nodes declared and initialised. Defaults to True.

## Exceptions

<i>NegativeValue</i>	beam_left_L_cl needs to be positive.
<i>NegativeValue</i>	beam_right_L_cl needs to be positive.
<i>NegativeValue</i>	col_top_L_cl needs to be positive.
<i>NegativeValue</i>	col_bottom_L_cl needs to be positive.
<i>NegativeValue</i>	depth_col needs to be positive.
<i>NegativeValue</i>	depth_beam needs to be positive.

## Returns

list: List with the nodes declared.

Definition at line 207 of file [GeometryTemplate.py](#).

```

00208     boundary_condition = True, show_plot = True):
00209     """
00210     Function that declares and initialises the grid nodes of an interior subassemblage. The panel zone
00211     geometry is defined by the two arguments
00212         depth_col and depth_beam.
00213
00213     @param beam_left_L_cl (float): Centerline length of the left beam (excluding the panel zone).
00214     @param beam_right_L_cl (float): Centerline length of the right beam (excluding the panle zone).
00215     @param col_top_L_cl (float): Centerline length of the top column (excluding the panel zone).
00216     @param col_bottom_L_cl (float): Centerline length of the bottom column (excluding the panel zone).
00217     @param depth_col (float): Depth of the columns for the panel zone.
00218     @param depth_beam (float): Depth of the beams for the panel zone.
00219     @param boundary_condition (bool, optional): Option to set already the boundary condition (bottom
00220     column pinned, beams fix only y movement).
00221     Defaults to True.
00221     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
00222     Defaults to True.
00222
00223     @exception NegativeValue: beam_left_L_cl needs to be positive.
00224     @exception NegativeValue: beam_right_L_cl needs to be positive.
00225     @exception NegativeValue: col_top_L_cl needs to be positive.
00226     @exception NegativeValue: col_bottom_L_cl needs to be positive.
00227     @exception NegativeValue: depth_col needs to be positive.
00228     @exception NegativeValue: depth_beam needs to be positive.
00229
00230     @returns list: List with the nodes declared.
00231     """
00232     # origin is the bottom left corner
00233     if beam_left_L_cl < 0: raise NegativeValue()
00234     if beam_right_L_cl < 0: raise NegativeValue()
00235     if col_top_L_cl < 0: raise NegativeValue()
00236     if col_bottom_L_cl < 0: raise NegativeValue()
00237     if depth_col < 0: raise NegativeValue()
00238     if depth_beam < 0: raise NegativeValue()
00239
00240     node(12, 0.0, col_bottom_L_cl+depth_beam/2)
00241     node(21, beam_left_L_cl+depth_col/2, 0.0)
00242     node(22, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam)
00243     node(23, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam+col_top_L_cl)

```

```

00244     node(32, beam_left_L_cl+depth_col+beam_right_L_cl, col_bottom_L_cl+depth_beam/2)
00245     node_array = [12, 21, 22, 23, 32]
00246
00247     if boundary_condition:
00248         fix(12, 0, 1, 0)
00249         fix(32, 0, 1, 0)
00250         fix(21, 1, 1, 0)
00251
00252     if show_plot:
00253         plot_nodes(node_array, "Subassemblage geometry template with only nodes", True)
00254         plt.grid()
00255
00256     return node_array
00257
00258
00259 # def DefineRCSSubassemblage():
00260 #     # WIP and experimental
00261
00262

```

### 6.8.2.4 Initialize2DModel()

```

def GeometryTemplate.Initialize2DModel (
    data_dir = "Results" )

```

Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.

#### Parameters

<i>data_dir</i>	(str, optional): Directory where the data will be stored. The function forces the user to define it just for good practice and consistency between projects. Defaults to "Results".
-----------------	---

Definition at line 25 of file [GeometryTemplate.py](#).

```

00025 def Initialize2DModel(data_dir = "Results"):
00026     """
00027     Function that initialise the project creating the 2D model with 3 DOF per node and set up a
    directory for the results.
00028
00029     @param data_dir (str, optional): Directory where the data will be stored.
00030     The function forces the user to define it just for good practice and consistency between
    projects.
00031     Defaults to "Results".
00032     """
00033     # Clear all
00034     wipe()
00035
00036     # Build model (2D - 3 DOF/node)
00037     model('basic', '-ndm', 2, '-ndf', 3)
00038
00039     # Main Results Folder
00040     if not os.path.exists(data_dir):
00041         os.makedirs(data_dir)
00042
00043

```

## 6.9 MaterialModels Namespace Reference

Module for the material models.

### Classes

- class [ConfMander1988Circ](#)

- Class that stores functions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.*
- class [ConfMander1988CircRCCircShape](#)  
*Class that is the children of [ConfMander1988Circ](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.*
  - class [ConfMander1988Rect](#)  
*Class that stores functions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.*
  - class [ConfMander1988RectRCRectShape](#)  
*Class that is the children of [ConfMander1988Rect](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.*
  - class [GMP1970](#)  
*Class that stores functions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.*
  - class [GMP1970RCRectShape](#)  
*Class that is the children of [GMP1970](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.*
  - class [Gupta1999](#)  
*Class that stores functions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.*
  - class [Gupta1999SteellShape](#)  
*Class that is the children of [Gupta1999](#) and combine the class [SteellShape](#) (section) to retrieve the information needed.*
  - class [MaterialModels](#)  
*Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.*
  - class [ModifiedIMK](#)  
*Class that stores functions and material properties of a steel double symmetric I-shape profile with modified Ibarra-↔ Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.*
  - class [ModifiedIMKSteellShape](#)  
*Class that is the children of [ModifiedIMK](#) and combine the class [SteellShape](#) (section) to retrieve the information needed.*
  - class [Skiadopoulos2021](#)  
*Class that stores functions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.*
  - class [Skiadopoulos2021RCS](#)  
*WIP: Class that is the children of [Skiadopoulos2021](#) and it's used for the panel zone spring in a RCS (RC column continuous, Steel beam).*
  - class [Skiadopoulos2021SteellShape](#)  
*Class that is the children of [Skiadopoulos2021](#) and combine the class [SteellShape](#) (section) to retrieve the information needed.*
  - class [UnconfMander1988](#)  
*Class that stores functions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.*
  - class [UnconfMander1988RCCircShape](#)  
*Class that is the children of [UnconfMander1988](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.*
  - class [UnconfMander1988RCRectShape](#)  
*Class that is the children of [UnconfMander1988](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.*
  - class [UniaxialBilinear](#)

Class that stores functions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

- class [UniaxialBilinearSteelShape](#)

Class that is the children of [UniaxialBilinear](#) and combine the class SteelShape (section) to retrieve the information needed.

- class [UVC](#)

Class that stores functions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCUniaxial.

- class [UVCCalibrated](#)

Class that is the children of [UVC](#) that retrieve calibrated parameters from UVC\_calibrated\_parameters.txt.

- class [UVCCalibratedRCCircShape](#)

Class that is the children of [UVCCalibrated](#) and combine the class RCCircShape (section) to retrieve the information needed.

- class [UVCCalibratedRCRectShape](#)

Class that is the children of [UVCCalibrated](#) and combines the class RCRectShape (section) to retrieve the information needed.

- class [UVCCalibratedSteelShapeFlange](#)

Class that is the children of [UVCCalibrated](#) and combine the class SteelShape (section) to retrieve the information needed for the material model of the flange (often used to the entire section).

- class [UVCCalibratedSteelShapeWeb](#)

Class that is the children of [UVCCalibrated](#) and combine the class SteelShape (section) to retrieve the information needed for the material model of the web.

## Functions

- def [Concrete01Funct](#) (fc, ec, fpcu, ecu, discretized\_eps)

Function with the equation of the curve of the Concrete01 model.

- def [Concrete04Funct](#) (fc, discretized\_eps, ec, Ec)

Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

- def [PlotConcrete01](#) (fc, ec, fpcu, ecu, ax, ID=0)

Function that plots the Concrete01 stress-strain curve.

- def [PlotConcrete04](#) (fc, Ec, ec, ecu, str Type, ax, ID=0)

Function that plots the confined/unconfined Concrete04 stress-strain curve.

### 6.9.1 Detailed Description

Module for the material models.

Carmine Schipani, 2021

### 6.9.2 Function Documentation

### 6.9.2.1 Concrete01Funct()

```
def MaterialModels.Concrete01Funct (
    fc,
    ec,
    fpcu,
    ecu,
    discretized_eps )
```

Function with the equation of the curve of the Concrete01 model.

For more information, see Kent-Scott-Park concrete material object with degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no tensile strength.

## Parameters

<i>fc</i>	(float): Compressive concrete yield stress (negative).
<i>ec</i>	(float): Compressive concrete yield strain (negative).
<i>fpcu</i>	(float): Concrete crushing strength (negative).
<i>ecu</i>	(float): Concrete strain at crushing strength (negative).
<i>discretized_eps</i>	(float): Variable strain.

## Returns

float: Stress in function of variable strain.

Definition at line 2969 of file [MaterialModels.py](#).

```

02969 def Concrete01Funcnt(fc, ec, fpcu, ecu, discretized_eps):
02970     """
02971     Function with the equation of the curve of the Concrete01 model.
02972     For more information, see Kent-Scott-Park concrete material object with
02973     degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no
02974     tensile strength.
02975     @param fc (float): Compressive concrete yield stress (negative).
02976     @param ec (float): Compressive concrete yield strain (negative).
02977     @param fpcu (float): Concrete crushing strength (negative).
02978     @param ecu (float): Concrete strain at crushing strength (negative).
02979     @param discretized_eps (float): Variable strain.
02980
02981     @returns float: Stress in function of variable strain.
02982     """
02983     if discretized_eps > ec:
02984         eta = discretized_eps/ec;
02985         return fc*(2*eta-eta*eta);
02986     else:
02987         Ttangent = (fc-fpcu)/(ec-ecu)
02988         return fc + Ttangent*(discretized_eps-ec);
02989
02990

```

## 6.9.2.2 Concrete04Funcnt()

```

def MaterialModels.Concrete04Funcnt (
    fc,
    discretized_eps,
    ec,
    Ec )

```

Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

## Parameters

<i>fc</i>	(float): Compressive concrete yield stress (negative).
<i>discretized_eps</i>	(float): Variable strain.
<i>ec</i>	(float): Compressive concrete yield strain (negative).
<i>Ec</i>	(float): Concrete Young modulus.

## Returns

float: Stress in function of variable strain.

Definition at line 2913 of file [MaterialModels.py](#).

```
02913 def Concrete04Funct(fc, discretized_eps, ec, Ec):
02914     """
02915     Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).
02916
02917     @param fc (float): Compressive concrete yield stress (negative).
02918     @param discretized_eps (float): Variable strain.
02919     @param ec (float): Compressive concrete yield strain (negative).
02920     @param Ec (float): Concrete Young modulus.
02921
02922     @returns float: Stress in function of variable strain.
02923     """
02924     x = discretized_eps/ec
02925     r = Ec / (Ec - fc/ec)
02926     return fc*x*r / (r-1+x*r)
02927
02928
```

### 6.9.2.3 PlotConcrete01()

```
def MaterialModels.PlotConcrete01 (
    fc,
    ec,
    fpcu,
    ecu,
    ax,
    ID = 0 )
```

Function that plots the Concrete01 stress-strain curve.

#### Parameters

<i>fc</i>	(float): Compressive concrete yield stress (negative).
<i>ec</i>	(float): Compressive concrete yield strain (negative).
<i>fpcu</i>	(float): Concrete crushing strength (negative).
<i>ecu</i>	(float): Concrete strain at crushing strength (negative).
<i>ax</i>	(matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
<i>ID</i>	(int, optional): ID of the material model. Defaults to 0 (= not defined).

Example: to create the plot, call this line to pass the correct ax: fig, ax = plt.subplots()

Definition at line 2991 of file [MaterialModels.py](#).

```
02991 def PlotConcrete01(fc, ec, fpcu, ecu, ax, ID = 0):
02992     """
02993     Function that plots the Concrete01 stress-strain curve.
02994
02995     @param fc (float): Compressive concrete yield stress (negative).
02996     @param ec (float): Compressive concrete yield strain (negative).
02997     @param fpcu (float): Concrete crushing strength (negative).
02998     @param ecu (float): Concrete strain at crushing strength (negative).
02999     @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
03000     @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).
03001
03002     Example: to create the plot, call this line to pass the correct ax:
03003     fig, ax = plt.subplots()
03004     """
03005
03006     # Data for plotting
03007     N = 1000
03008     x_axis = np.zeros(N)
03009     y_axis = np.zeros(N)
03010     for i in range(N):
03011         x_axis[i] = i/N*ecu
03012         y_axis[i] = Concrete01Funct(fc, ec, fpcu, ecu, x_axis[i])
03013
```



```

03014
03015     ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k--', label = "Co01")
03016     ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
03017           title='Mander 1988 (Concrete01) material model (ID={})'.format(ID))
03018     plt.legend()
03019     plt.grid()
03020
03021
03022 # Private functions

```

#### 6.9.2.4 PlotConcrete04()

```

def MaterialModels.PlotConcrete04 (
    fc,
    Ec,
    ec,
    ecu,
    str Type,
    ax,
    ID = 0 )

```

Function that plots the confined/unconfined Concrete04 stress-strain curve.

##### Parameters

<i>fc</i>	(float): Compressive concrete yield strength (needs to be negative).
<i>Ec</i>	(float): Young modulus.
<i>ec</i>	(float): Compressive concrete yield strain.
<i>ecu</i>	(float): Compressive concrete failure strain (negative).
<i>Type</i>	(str): Type of concrete (confined = 'C', unconfined = 'U')
<i>ax</i>	(matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
<i>ID</i>	(int, optional): ID of the material model. Defaults to 0 (= not defined).

##### Exceptions

<i>NameError</i>	
------------------	--

Example: to create the plot, call this line to pass the correct ax: fig, ax = plt.subplots()

Definition at line 2929 of file [MaterialModels.py](#).

```

02929 def PlotConcrete04(fc, Ec, ec, ecu, Type: str, ax, ID = 0):
02930     """
02931     Function that plots the confined/unconfined Concrete04 stress-strain curve.
02932
02933     @param fc (float): Compressive concrete yield strength (needs to be negative).
02934     @param Ec (float): Young modulus.
02935     @param ec (float): Compressive concrete yield strain.
02936     @param ecu (float): Compressive concrete failure strain (negative).
02937     @param Type (str): Type of concrete (confined = 'C', unconfined = 'U')
02938     @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
02939     @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).
02940
02941     @exception NameError:
02942
02943     Example: to create the plot, call this line to pass the correct ax:
02944         fig, ax = plt.subplots()
02945     """
02946     if Type == "C":
02947         name = "Confined (Co04)"
02948     elif Type == "U":

```

```

02949         name = "Unconfined (Co04)"
02950     else:
02951         raise NameError("Type should be C or U (ID={})".format(ID))
02952
02953     # Data for plotting
02954     N = 1000
02955     x_axis = np.zeros(N)
02956     y_axis = np.zeros(N)
02957     for i in range(N):
02958         x_axis[i] = i/N*ecu
02959         y_axis[i] = Concrete04Funct(fc, x_axis[i], ec, Ec)
02960
02961
02962     ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k-', label = name)
02963     ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
02964           title='Mander 1988 (Concrete04) material model (ID={})'.format(ID))
02965     plt.legend()
02966     plt.grid()
02967
02968

```

## 6.10 MemberModel Namespace Reference

Module for the member model.

### Classes

- class [ElasticElement](#)  
*Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.*
- class [ElasticElementSteelShape](#)  
*Class that is the children of [ElasticElement](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.*
- class [ForceBasedElement](#)  
*Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.*
- class [ForceBasedElementFibersCircRCCircShape](#)  
*Class that is the children of [ForceBasedElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.*
- class [ForceBasedElementFibersIShapeSteelShape](#)  
*Class that is the children of [ForceBasedElement](#) and combine the class [FibersIShapeSteelShape](#) (fiber section) to retrieve the information needed.*
- class [ForceBasedElementFibersRectRCRectShape](#)  
*Class that is the children of [ForceBasedElement](#) and combine the class [FibersRectRCRectShape](#) (fiber section) to retrieve the information needed.*
- class [GIFBElement](#)  
*Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.*
- class [GIFBElementFibersCircRCCircShape](#)  
*Class that is the children of [GIFBElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.*
- class [GIFBElementFibersRectRCRectShape](#)  
*Class that is the children of [GIFBElement](#) and combine the class [FibersRectRCRectShape](#) (fiber section) to retrieve the information needed.*
- class [GIFBElementRCCircShape](#)  
*Class that is the children of [GIFBElement](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.*
- class [GIFBElementRCRectShape](#)

Class that is the children of [GIFBElement](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.

- class [MemberModel](#)

Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class [PanelZone](#)

Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class [PanelZoneRCS](#)

WIP: Class that is the children of [PanelZone](#) and it's used for the panel zone in a RCS (RC column continuous, Steel beam).

- class [PanelZoneSteelShape](#)

Class that is the children of [PanelZone](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

- class [PanelZoneSteelShapeGupta1999](#)

Class that is the children of [PanelZoneSteelShape](#) and automatically create the spring material model Gupta 1999 (ID = master\_node\_ID).

- class [PanelZoneSteelShapeSkiadopoulos2021](#)

Class that is the children of [PanelZoneSteelShape](#) and automatically create the spring material model Skiadopoulos 2021 (ID = master\_node\_ID).

- class [SpringBasedElement](#)

Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class [SpringBasedElementModifiedIMKSteelShape](#)

Class that is the children of [SpringBasedElement](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

- class [SpringBasedElementSteelShape](#)

Class that is the children of [SpringBasedElement](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

## Functions

- def [DefinePanelZoneElements](#) (MasterNodeID, E, RigidA, RigidI, TransfID)

Function that defines the 8 panel zone elements.

- def [DefinePanelZoneNodes](#) (int MasterNodeID, MidPanelZoneWidth, MidPanelZoneHeight)

Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).

### 6.10.1 Detailed Description

Module for the member model.

Carmine Schipani, 2021

### 6.10.2 Function Documentation

### 6.10.2.1 DefinePanelZoneElements()

```
def MemberModel.DefinePanelZoneElements (
    MasterNodeID,
    E,
    RigidA,
    RigidI,
    TransfID )
```

Function that defines the 8 panel zone elements.

For the ID convention, see DefinePanelZoneNodes.

#### Parameters

<i>MasterNodeID</i>	(int): ID of the master node (central top node that should be a grid node).
<i>E</i>	(float): Young modulus.
<i>RigidA</i>	(float): A very rigid area.
<i>RigidI</i>	(float): A very rigid moment of inertia.
<i>TransfID</i>	(int): The geometric transformation (for more information, see OpenSeesPy documentation).

#### Returns

list: List of lists, with each list containing the ID of the element, of node i and node j.

Definition at line 432 of file [MemberModel.py](#).

```
00432 def DefinePanelZoneElements(MasterNodeID, E, RigidA, RigidI, TransfID):
00433     """
00434     Function that defines the 8 panel zone elements. For the ID convention, see DefinePanelZoneNodes.
00435
00436     @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
00437     @param E (float): Young modulus.
00438     @param RigidA (float): A very rigid area.
00439     @param RigidI (float): A very rigid moment of inertia.
00440     @param TransfID (int): The geometric transformation (for more information, see OpenSeesPy
00441     documentation).
00442
00443     @returns list: List of lists, with each list containing the ID of the element, of node i and node
00444     j.
00445     """
00446     # Compute the ID of the nodes obeying to the convention used
00447     xy = MasterNodeID
00448     xy1 = IDConvention(xy, 1)
00449     xy01 = IDConvention(xy, 1, 1)
00450     xy02 = IDConvention(xy, 2, 1)
00451     xy03 = IDConvention(xy, 3, 1)
00452     xy04 = IDConvention(xy, 4, 1)
00453     xy05 = IDConvention(xy, 5, 1)
00454     xy06 = IDConvention(xy, 6, 1)
00455     xy07 = IDConvention(xy, 7, 1)
00456     xy08 = IDConvention(xy, 8, 1)
00457     xy09 = IDConvention(xy, 9, 1)
00458     xy10 = IDConvention(xy, 10)
00459
00460     # Create element IDs using the convention: xy(a)xy(a) with xy(a) = NodeID i and j
00461     # Starting at MasterNodeID, clockwise
00462     # if MasterNodeID > 99:
00463     #     print("Warning, convention: MasterNodeID's digits should be 2")
00464
00465     ele1 = IDConvention(xy, xy1)
00466     ele2 = IDConvention(xy01, xy02)
00467     ele3 = IDConvention(xy02, xy03)
00468     ele4 = IDConvention(xy04, xy05)
00469     ele5 = IDConvention(xy05, xy06)
00470     ele6 = IDConvention(xy07, xy08)
00471     ele7 = IDConvention(xy08, xy09)
00472     ele8 = IDConvention(xy10, xy)
00473
00474     # Create panel zone elements
```

```

00473     # ID ndI ndJ A E I Transf
00474     element("elasticBeamColumn", ele1, xy, xyl, RigidA, E, RigidI, TransfID)
00475     element("elasticBeamColumn", ele2, xy01, xy02, RigidA, E, RigidI, TransfID)
00476     element("elasticBeamColumn", ele3, xy02, xy03, RigidA, E, RigidI, TransfID)
00477     element("elasticBeamColumn", ele4, xy04, xy05, RigidA, E, RigidI, TransfID)
00478     element("elasticBeamColumn", ele5, xy05, xy06, RigidA, E, RigidI, TransfID)
00479     element("elasticBeamColumn", ele6, xy07, xy08, RigidA, E, RigidI, TransfID)
00480     element("elasticBeamColumn", ele7, xy08, xy09, RigidA, E, RigidI, TransfID)
00481     element("elasticBeamColumn", ele8, xy10, xy, RigidA, E, RigidI, TransfID)
00482
00483     # Create element array for further manipulations
00484     element_array = [[ele1, xy, xyl],
00485                     [ele2, xy01, xy02],
00486                     [ele3, xy02, xy03],
00487                     [ele4, xy04, xy05],
00488                     [ele5, xy05, xy06],
00489                     [ele6, xy07, xy08],
00490                     [ele7, xy08, xy09],
00491                     [ele8, xy10, xy]]
00492
00493     return element_array
00494
00495

```

### 6.10.2.2 DefinePanelZoneNodes()

```

def MemberModel.DefinePanelZoneNodes (
    int MasterNodeID,
    MidPanelZoneWidth,
    MidPanelZoneHeight )

```

Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).

ID convention for the panel zone:

PZNodeID: 12 nodes: top right 1xy (master), 1xy1 top right, 1xy09,1xy10 1xy 1xy1,1xy01

clockwise 10 nodes xy01-xy10 (with double node at corners) o-----o-----o

Spring at node 1xy1 | |

PZElementID: 8 elements: starting at node 1xy, clockwise | |

(see function DefinePanelZoneElements for more info) | |

```

| |
1xy08 o o 1xy02

```

```

| |
| |
| |
| |

```

```

o-----o-----o

```

1xy06,1xy07 1xy05 1xy03,1xy04

Note that the top right node is defined differently because is where the spring is.

#### Parameters

<i>MasterNodeID</i>	(int): ID of the master node (central top node that should be a grid node).
<i>MidPanelZoneWidth</i>	(float): Mid panel zone width.
<i>MidPanelZoneHeight</i>	(float): Mid panel zone height.

Definition at line 388 of file [MemberModel.py](#).

```

00388 def DefinePanelZoneNodes(MasterNodeID: int, MidPanelZoneWidth, MidPanelZoneHeight):
00389     """

```

```

00390     Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master
00391     node (top center one).
00392     ID convention for the panel zone: \n
00393     PZNodeID:      12 nodes: top right lxy (master), lxy1 top right,
00394     lxy09,lxy10    lxy    lxy1,lxy01 \n
00395     clockwise 10 nodes xy01-xy10 (with double node at corners)
00396     o-----o-----o \n
00397     |           |           | \n
00398     |           |           | \n
00399     |           |           | \n
00400     |           |           | \n
00401     |           |           | \n
00402     |           |           | \n
00403     |           |           | \n
00404     o-----o-----o \n
00405     lxy06,lxy07    lxy05    lxy03,lxy04 \n
00406     Note that the top right node is defined differently because is where the spring is.
00407     @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
00408     @param MidPanelZoneWidth (float): Mid panel zone width.
00409     @param MidPanelZoneHeight (float): Mid panel zone height.
00410     """
00411     # Get node coord and define useful variables
00412     m_node = np.array(nodeCoord(MasterNodeID))
00413     AxisCL = m_node[0]
00414     FloorCL = m_node[1] - MidPanelZoneHeight
00415     # Convention: Node of the spring (top right) is xyl
00416     node(IDConvention(MasterNodeID, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
00417     # Convention: Two notes in the corners (already defined one, xyl) clockwise from xy01 to xyl0
00418     node(IDConvention(MasterNodeID, 1, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
00419     node(IDConvention(MasterNodeID, 2, 1), AxisCL+MidPanelZoneWidth, FloorCL)
00420     node(IDConvention(MasterNodeID, 3, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
00421     node(IDConvention(MasterNodeID, 4, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
00422     node(IDConvention(MasterNodeID, 5, 1), AxisCL, FloorCL-MidPanelZoneHeight)
00423     node(IDConvention(MasterNodeID, 6, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
00424     node(IDConvention(MasterNodeID, 7, 1), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
00425     node(IDConvention(MasterNodeID, 8, 1), AxisCL-MidPanelZoneWidth, FloorCL)
00426     node(IDConvention(MasterNodeID, 9, 1), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
00427     node(IDConvention(MasterNodeID, 10), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
00428     """
00429     """
00430     """
00431     """

```

## 6.11 Section Namespace Reference

Module for the section (steel I shape profiles, RC circular/square/rectangular sections).

### Classes

- class [RCCircShape](#)  
Class that stores functions, geometric and mechanical properties of RC circular shape profile.
- class [RCRectShape](#)  
Class that stores functions, geometric and mechanical properties of RC rectangular shape profile.
- class [RCSquareShape](#)  
Class that is the children of [RCRectShape](#) and cover the specific case of square RC sections.
- class [Section](#)  
Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).
- class [SteelShape](#)  
Class that stores functions, geometric and mechanical properties of a steel double symmetric I-shape profile.

## Functions

- def [ComputeACircle](#) (D)  
*Function that computes the area of one circle (reinforcing bar or hoop).*
- def [ComputeRho](#) (A, nr, A\_tot)  
*Compute the ratio of area of a reinforcement to area of a section.*

### 6.11.1 Detailed Description

Module for the section (steel I shape profiles, RC circular/square/rectangular sections).

Carmine Schipani, 2021

### 6.11.2 Function Documentation

#### 6.11.2.1 [ComputeACircle\(\)](#)

```
def Section.ComputeACircle (
    D )
```

Function that computes the area of one circle (reinforcing bar or hoop).

##### Parameters

<i>D</i>	(float): Diameter of the circle (reinforcing bar of hoop).
----------	--

##### Returns

float: Area the circle (for reinforcing bars or hoops).

Definition at line 763 of file [Section.py](#).

```
00763 def ComputeACircle(D):
00764     """
00765     Function that computes the area of one circle (reinforcing bar or hoop).
00766
00767     @param D (float): Diameter of the circle (reinforcing bar of hoop).
00768
00769     @returns float: Area the circle (for reinforcing bars or hoops).
00770     """
00771     return D**2/4.0*math.pi
00772
00773
```

#### 6.11.2.2 [ComputeRho\(\)](#)

```
def Section.ComputeRho (
    A,
    nr,
    A_tot )
```

Compute the ratio of area of a reinforcement to area of a section.

## Parameters

<i>A</i>	(float): Area of reinforcement.
<i>nr</i>	(float): Number of reinforcement (allow float for computing ratio with different area; just convert the other areas to one and compute the equivalent n).
<i>A_tot</i>	(float): Area of the concrete.

## Returns

float: Ratio.

Definition at line 774 of file [Section.py](#).

```

00774 def ComputeRho(A, nr, A_tot):
00775     """
00776     Compute the ratio of area of a reinforcement to area of a section.
00777
00778     @param A (float): Area of reinforcement.
00779     @param nr (float): Number of reinforcement (allow float for computing ratio with different area;
00780                     just convert the other areas to one and compute the equivalent n).
00781     @param A_tot (float): Area of the concrete.
00782
00783     @returns float: Ratio.
00784     """
00785     return nr * A / A_tot

```

## 6.12 Units Namespace Reference

Module with the units conversion and the definition of the units used as default (m, N, s).

## Variables

- float `cm2_unit` = `cm_unit*cm_unit`
- float `cm3_unit` = `cm_unit*cm_unit*cm_unit`
- float `cm4_unit` = `cm3_unit*cm_unit`
- float `cm_unit` = `m_unit*1e-2`
- float `dm2_unit` = `dm_unit*dm_unit`
- float `dm3_unit` = `dm_unit*dm_unit*dm_unit`
- float `dm4_unit` = `dm3_unit*dm_unit`
- float `dm_unit` = `m_unit*1e-1`
- string `force_unit` = "N"
- float `ft2_unit` = `ft_unit*ft_unit`
- float `ft3_unit` = `ft_unit*ft_unit*ft_unit`
- float `ft4_unit` = `ft3_unit*ft_unit`
- float `ft_unit` = `m_unit*0.3048`
- float `GN_unit` = `N_unit*1e9`
- float `GPa_unit` = `Pa_unit*1e9`
- float `hours_unit` = `min_unit*60`
- float `inch2_unit` = `inch_unit*inch_unit`
- float `inch3_unit` = `inch_unit*inch_unit*inch_unit`
- float `inch4_unit` = `inch3_unit*inch_unit`
- float `inch_unit` = `m_unit*0.0254`
- float `kg_unit` = `N_unit*s_unit**2/m_unit`
- float `kip_unit` = `N_unit*4448.2216`
- float `km_unit` = `m_unit*1e3`



- float `kN_unit` = `N_unit*1e3`
- float `kNm_unit` = `kN_unit*m_unit`
- float `kNmm_unit` = `kN_unit*mm_unit`
- float `kPa_unit` = `Pa_unit*1e3`
- float `ksi_unit` = `psi_unit*1000`
- string `length_unit` = "m"
- float `m2_unit` = `m_unit*m_unit`
- float `m3_unit` = `m_unit*m_unit*m_unit`
- float `m4_unit` = `m3_unit*m_unit`
- float `m_unit` = 1.0
- float `mile_unit` = `m_unit*1609.34`
- float `min_unit` = `s_unit*60`
- float `mm2_unit` = `mm_unit*mm_unit`
- float `mm3_unit` = `mm_unit*mm_unit*mm_unit`
- float `mm4_unit` = `mm3_unit*mm_unit`
- float `mm_unit` = `m_unit*1e-3`
- float `MN_unit` = `N_unit*1e6`
- float `MNm_unit` = `MN_unit*m_unit`
- float `MNmm_unit` = `MN_unit*mm_unit`
- float `MPa_unit` = `Pa_unit*1e6`
- float `N_unit` = 1.0
- float `Nm_unit` = `N_unit*m_unit`
- float `Nmm_unit` = `N_unit*mm_unit`
- float `Pa_unit` = `N_unit/m2_unit`
- float `pound_unit` = `kg_unit*0.45359237`
- float `psi_unit` = `Pa_unit*6894.76`
- float `s_unit` = 1.0
- float `t_unit` = `kg_unit*1e3`
- string `time_unit` = "s"

### 6.12.1 Detailed Description

Module with the units conversion and the definition of the units used as default (m, N, s).

Note that the decision of which unit for each measure (distance, force, mass, time) is equal to 1 is not arbitrary: for example the natural frequency is computed behind the scene by the OpenSeesPy framework, thus the stiffness of the structure divided by the mass should result in a unit of 1 (thus seconds).

Furthermore, there are constants like the gravitational one *g* that is dependent on this decision. If the units are used in a consistent way (using this library), these issues can be avoided.

Carmine Schipani, 2021

### 6.12.2 Variable Documentation

#### 6.12.2.1 `cm2_unit`

```
float cm2_unit = cm_unit*cm_unit
```

Definition at line 28 of file [Units.py](#).

#### 6.12.2.2 cm3\_unit

```
float cm3_unit = cm_unit*cm_unit*cm_unit
```

Definition at line 36 of file [Units.py](#).

#### 6.12.2.3 cm4\_unit

```
float cm4_unit = cm3_unit*cm_unit
```

Definition at line 44 of file [Units.py](#).

#### 6.12.2.4 cm\_unit

```
float cm_unit = m_unit*1e-2
```

Definition at line 19 of file [Units.py](#).

#### 6.12.2.5 dm2\_unit

```
float dm2_unit = dm_unit*dm_unit
```

Definition at line 29 of file [Units.py](#).

#### 6.12.2.6 dm3\_unit

```
float dm3_unit = dm_unit*dm_unit*dm_unit
```

Definition at line 37 of file [Units.py](#).

#### 6.12.2.7 dm4\_unit

```
float dm4_unit = dm3_unit*dm_unit
```

Definition at line 45 of file [Units.py](#).

#### 6.12.2.8 dm\_unit

```
float dm_unit = m_unit*1e-1
```

Definition at line 20 of file [Units.py](#).

#### 6.12.2.9 force\_unit

```
string force_unit = "N"
```

Definition at line 13 of file [Units.py](#).

#### 6.12.2.10 ft2\_unit

```
float ft2_unit = ft_unit*ft_unit
```

Definition at line 32 of file [Units.py](#).

#### 6.12.2.11 ft3\_unit

```
float ft3_unit = ft_unit*ft_unit*ft_unit
```

Definition at line 40 of file [Units.py](#).

#### 6.12.2.12 ft4\_unit

```
float ft4_unit = ft3_unit*ft_unit
```

Definition at line 48 of file [Units.py](#).

#### 6.12.2.13 ft\_unit

```
float ft_unit = m_unit*0.3048
```

Definition at line 23 of file [Units.py](#).

#### 6.12.2.14 GN\_unit

```
float GN_unit = N_unit*1e9
```

Definition at line 53 of file [Units.py](#).

#### 6.12.2.15 GPa\_unit

```
float GPa_unit = Pa_unit*1e9
```

Definition at line 73 of file [Units.py](#).

#### 6.12.2.16 hours\_unit

```
float hours_unit = min_unit*60
```

Definition at line 79 of file [Units.py](#).

#### 6.12.2.17 inch2\_unit

```
float inch2_unit = inch_unit*inch_unit
```

Definition at line 31 of file [Units.py](#).

#### 6.12.2.18 inch3\_unit

```
float inch3_unit = inch_unit*inch_unit*inch_unit
```

Definition at line 39 of file [Units.py](#).

#### 6.12.2.19 inch4\_unit

```
float inch4_unit = inch3_unit*inch_unit
```

Definition at line 47 of file [Units.py](#).

#### 6.12.2.20 inch\_unit

```
float inch_unit = m_unit*0.0254
```

Definition at line 22 of file [Units.py](#).

#### 6.12.2.21 kg\_unit

```
float kg_unit = N_unit*s_unit**2/m_unit
```

Definition at line 65 of file [Units.py](#).

#### 6.12.2.22 kip\_unit

```
float kip_unit = N_unit*4448.2216
```

Definition at line 54 of file [Units.py](#).

#### 6.12.2.23 km\_unit

```
float km_unit = m_unit*1e3
```

Definition at line 21 of file [Units.py](#).

#### 6.12.2.24 kN\_unit

```
float kN_unit = N_unit*1e3
```

Definition at line 51 of file [Units.py](#).

#### 6.12.2.25 kNm\_unit

```
float kNm_unit = kN_unit*m_unit
```

Definition at line 58 of file [Units.py](#).

#### 6.12.2.26 kNmm\_unit

```
float kNmm_unit = kN_unit*mm_unit
```

Definition at line 61 of file [Units.py](#).

#### 6.12.2.27 kPa\_unit

```
float kPa_unit = Pa_unit*1e3
```

Definition at line 71 of file [Units.py](#).

#### 6.12.2.28 ksi\_unit

```
float ksi_unit = psi_unit*1000
```

Definition at line 75 of file [Units.py](#).

#### 6.12.2.29 length\_unit

```
string length_unit = "m"
```

Definition at line 11 of file [Units.py](#).

#### 6.12.2.30 m2\_unit

```
float m2_unit = m_unit*m_unit
```

Definition at line 30 of file [Units.py](#).

#### 6.12.2.31 m3\_unit

```
float m3_unit = m_unit*m_unit*m_unit
```

Definition at line 38 of file [Units.py](#).

#### 6.12.2.32 m4\_unit

```
float m4_unit = m3_unit*m_unit
```

Definition at line 46 of file [Units.py](#).

#### 6.12.2.33 m\_unit

```
float m_unit = 1.0
```

Definition at line 10 of file [Units.py](#).

#### 6.12.2.34 mile\_unit

```
float mile_unit = m_unit*1609.34
```

Definition at line 24 of file [Units.py](#).

#### 6.12.2.35 min\_unit

```
float min_unit = s_unit*60
```

Definition at line 78 of file [Units.py](#).

#### 6.12.2.36 mm2\_unit

```
float mm2_unit = mm_unit*mm_unit
```

Definition at line 27 of file [Units.py](#).

#### 6.12.2.37 mm3\_unit

```
float mm3_unit = mm_unit*mm_unit*mm_unit
```

Definition at line 35 of file [Units.py](#).

#### 6.12.2.38 mm4\_unit

```
float mm4_unit = mm3_unit*mm_unit
```

Definition at line 43 of file [Units.py](#).

#### 6.12.2.39 mm\_unit

```
float mm_unit = m_unit*1e-3
```

Definition at line 18 of file [Units.py](#).

#### 6.12.2.40 MN\_unit

```
float MN_unit = N_unit*1e6
```

Definition at line 52 of file [Units.py](#).

#### 6.12.2.41 MNm\_unit

```
float MNm_unit = MN_unit*m_unit
```

Definition at line 59 of file [Units.py](#).

#### 6.12.2.42 MNmm\_unit

```
float MNmm_unit = MN_unit*mm_unit
```

Definition at line 62 of file [Units.py](#).

#### 6.12.2.43 MPa\_unit

```
float MPa_unit = Pa_unit*1e6
```

Definition at line 72 of file [Units.py](#).



#### 6.12.2.44 N\_unit

```
float N_unit = 1.0
```

Definition at line 12 of file [Units.py](#).

#### 6.12.2.45 Nm\_unit

```
float Nm_unit = N_unit*m_unit
```

Definition at line 57 of file [Units.py](#).

#### 6.12.2.46 Nmm\_unit

```
float Nmm_unit = N_unit*mm_unit
```

Definition at line 60 of file [Units.py](#).

#### 6.12.2.47 Pa\_unit

```
float Pa_unit = N_unit/m2_unit
```

Definition at line 70 of file [Units.py](#).

#### 6.12.2.48 pound\_unit

```
float pound_unit = kg_unit*0.45359237
```

Definition at line 67 of file [Units.py](#).

#### 6.12.2.49 psi\_unit

```
float psi_unit = Pa_unit*6894.76
```

Definition at line 74 of file [Units.py](#).

**6.12.2.50 s\_unit**

```
float s_unit = 1.0
```

Definition at line 14 of file [Units.py](#).

**6.12.2.51 t\_unit**

```
float t_unit = kg_unit*1e3
```

Definition at line 66 of file [Units.py](#).

**6.12.2.52 time\_unit**

```
string time_unit = "s"
```

Definition at line 15 of file [Units.py](#).

## Chapter 7

# Class Documentation

### 7.1 Analysis Class Reference

Class dedicated to the analysis of the OpenSeesPy model.

#### Public Member Functions

- `def __init__ (self, str data\_dir, str name\_ODB, algo="KrylovNewton", test_type="NormDisplncr", test_opt=0, max_iter=MAX_ITER, tol=TOL, allow_smaller_step=False)`  
*The constructor of the class.*
- `def DeformedShape (self, scale=1, animate=False, dt=0.01)`  
*Method that shows the final deformed shape of the model.*
- `def FiberResponse (self, int ele_fiber_ID_analysed, fiber_section=1, animate_stress=False, animate_strain=False, fps=25)`  
*Method that shows the final stress response of the fiber section chosen.*
- `def Gravity (self, list loaded_nodes, list Fy, int timeSeries_ID, int pattern_ID, n_step=10, timeSeries_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_type="BandGeneral", analysis_type="Static", show_plot=False)`  
*Method to perform the gravity analysis with vertical loadings (load-control).*
- `def LateralForce (self, list loaded_nodes, list Fx, int timeSeries_ID, int pattern_ID, n_step=1000, fiber_ID_analysed=-1, fiber_section=1, timeSeries_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_type="BandGeneral", analysis_type="Static", show_plot=True, block=False)`  
*Method to perform the lateral force analysis with lateral loading (load-control).*
- `def LoadingProtocol (self, int CtrlNode, np.ndarray discr_LP, int timeSeries_ID, int pattern_ID, Fx=1 *kN_unit, ele_fiber_ID_analysed=-1, fiber_section=1, timeSeries_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_type="UmfPack", analysis_type="Static", show_plot=True, block=False)`  
*Method to perform a loading protocol analysis (displacement-control).*
- `def Pushover (self, int CtrlNode, Dmax, Dincr, int timeSeries_ID, int pattern_ID, Fx=1 *kN_unit, ele_fiber_ID_analysed=-1, fiber_section=1, timeSeries_type="Linear", pattern_type="Plain", constraints_type="Plain", numberer_type="RCM", system_type="UmfPack", analysis_type="Static", show_plot=True, block=False)`  
*Method to perform a pushover analysis (displacement-control).*

## Public Attributes

- [algo](#)
- [allow\\_smaller\\_step](#)
- [data\\_dir](#)
- [load\\_case](#)
- [max\\_iter](#)
- [name\\_ODB](#)
- [test\\_opt](#)
- [test\\_type](#)
- [tol](#)

### 7.1.1 Detailed Description

Class dedicated to the analysis of the OpenSeesPy model.

The Gravity method should be run first to perform the Load-control analysis (apply the vertical load). If no vertical load, this method can be omitted.

Then only one of the Displacement-control (Pushover or LoadingProtocol) or Load-control (LateralForce) analysis can ran.

After the analysis reach convergence in the last step, for the postprocessing, the DeformedShape method can be used to see the final deformed shape and the animation of the entire loading protocol; the FiberResponse method can be used to see the animation of the same fiber section recorded during the analysis (strain and/or stress).

Definition at line 16 of file [AnalysisAndPostProcessing.py](#).

### 7.1.2 Constructor & Destructor Documentation

#### 7.1.2.1 `__init__()`

```
def __init__ (
    self,
    str data_dir,
    str name_ODB,
    algo = "KrylovNewton",
    test_type = "NormDispIncr",
    test_opt = 0,
    max_iter = MAX_ITER,
    tol = TOL,
    allow_smaller_step = False )
```

The constructor of the class.

#### Parameters

<i>data_dir</i>	(str): Directory in which the results from the analysis will be stored. Use the recorders (from OpenSeesPy) or the Record method from <a href="#">MemberModel</a> .
<i>name_ODB</i>	(str): Name for the folder in which the data for the animations and the fibers are stored.

## Parameters

<i>algo</i>	(str, optional): Type of algorithm chosen for the analysis. It determines how to construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear equation. For more information on the available types, see the OpenSeesPy documentation. Defaults to "KrylovNewton".
<i>test_type</i>	(str, optional): Type of test chosen for the analysis. It determines how to construct a ConvergenceTest object. Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence has been achieved at the end of an iteration step. For more information on the available types, see the OpenSeesPy documentation. Defaults to "NormDispIncr".
<i>test_opt</i>	(int, optional): Print-flag from 0 to 5 used to receive more info during the iteration (for example: 0 print nothing and 2 print information on norms and number of iterations at end of successful test). For more information, see the OpenSeesPy documentation. Defaults to 0.
<i>max_iter</i>	(float, optional): Maximal number of iterations to check. Defaults to MAX_ITER (from Constants Module).
<i>tol</i>	(float, optional): Tolerance criteria used to check for convergence. Defaults to TOL (from Constants Module).
<i>allow_smaller_step</i>	(bool, optional): Allow smaller steps in the displacement-control analysis. Defaults to False.

## Exceptions

<i>NegativeValue</i>	The argument max_iter should be positive.
<i>NegativeValue</i>	The argument tol should be positive.

Definition at line 22 of file [AnalysisAndPostProcessing.py](#).

```

00022     def __init__(self, data_dir: str, name_ODB: str, algo = "KrylovNewton", test_type =
    "NormDispIncr", test_opt = 0, max_iter = MAX_ITER, tol = TOL, allow_smaller_step = False):
00023         """
00024         The constructor of the class.
00025
00026         @param data_dir (str): Directory in which the results from the analysis will be stored. Use
    the recorders (from OpenSeesPy) or the Record method from MemberModel.
00027         @param name_ODB (str): Name for the folder in which the data for the animations and the fibers
    are stored.
00028         @param algo (str, optional): Type of algorithm chosen for the analysis. It determines how to
    construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the
    non-linear equation.
00029         For more information on the available types, see the OpenSeesPy documentation. Defaults to
    "KrylovNewton".
00030         @param test_type (str, optional): Type of test chosen for the analysis. It determines how to
    construct a ConvergenceTest object.
00031         Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if
    convergence has been achieved at the end of an iteration step.
00032         For more information on the available types, see the OpenSeesPy documentation. Defaults to
    "NormDispIncr".
00033         @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the
    iteration
00034         (for example: 0 print nothing and 2 print information on norms and number of iterations at
    end of successful test).
00035         For more information, see the OpenSeesPy documentation. Defaults to 0.
00036         @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER
    (from Constants Module).
00037         @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to
    TOL (from Constants Module).
00038         @param allow_smaller_step (bool, optional): Allow smaller steps in the displacement-control
    analysis. Defaults to False.
00039
00040         @exception NegativeValue: The argument max_iter should be positive.
00041         @exception NegativeValue: The argument tol should be positive.
00042         """
00043         if max_iter < 0: raise NegativeValue()
00044         if tol < 0: raise NegativeValue()
00045         if not os.path.exists(data_dir):
00046             print("Folder {} not found in this directory; creating one".format(data_dir))
00047             os.makedirs(data_dir)

```

```

00048
00049         self.data_dir = data_dir
00050         self.name_ODB = name_ODB
00051         self.algo = algo
00052         self.test_type = test_type
00053         self.tol = tol
00054         self.test_opt = test_opt
00055         self.max_iter = max_iter
00056         self.allow_smaller_step = allow_smaller_step
00057         self.load_case = "None"
00058
00059

```

## 7.1.3 Member Function Documentation

### 7.1.3.1 DeformedShape()

```

def DeformedShape (
    self,
    scale = 1,
    animate = False,
    dt = 0.01 )

```

Method that shows the final deformed shape of the model.

It can also show the animation that shows how the model behaved during the analysis.

#### Parameters

<i>scale</i>	(int, optional): The scaling factor to magnify the deformation. The value should be adjusted for each model. Defaults to 1.
<i>animate</i>	(bool, optional): Option to show the animation of the model during the analysis. Defaults to False.
<i>dt</i>	(float, optional): The time step between every iteration. Defaults to 0.01.

#### Exceptions

<i>NameError</i>	The methods for the analysis were not called.
------------------	---

Definition at line 652 of file [AnalysisAndPostProcessing.py](#).

```

00652     def DeformedShape(self, scale = 1, animate = False, dt = 0.01):
00653         """
00654         Method that shows the final deformed shape of the model. It can also show the animation that
00655         shows how the model behaved during the analysis.
00656
00657         @param scale (int, optional): The scaling factor to magnify the deformation. The value should
00658         be adjusted for each model. Defaults to 1.
00659         @param animate (bool, optional): Option to show the animation of the model during the
00660         analysis. Defaults to False.
00661         @param dt (float, optional): The time step between every iteration. Defaults to 0.01.
00662
00663         @exception NameError: The methods for the analysis were not called.
00664         """
00665         if self.load_case == "None": raise NameError("The analysis is not complete.")
00666
00667         # Display deformed shape, the scaling factor needs to be adjusted for each model
00668         opsplt.plot_deformedshape(Model = self.name_ODB, LoadCase=self.load_case, scale = scale)
00669         if animate:
00670             opsplt.animate_deformedshape(Model = self.name_ODB, LoadCase=self.load_case, dt = dt,
00671             scale = scale)
00672
00673

```

00669

### 7.1.3.2 FiberResponse()

```
def FiberResponse (
    self,
    int ele_fiber_ID_analysed,
    fiber_section = 1,
    animate_stress = False,
    animate_strain = False,
    fps = 25 )
```

Method that shows the final stress response of the fiber section chosen.

It can also show the animation that shows how the fiber section behaved during the analysis. The fiber ID and section needs to be recorded during the analysis, thus if the method LateralForce, Pushover or LoadingProtocol was used, the same fiber ID and section need to be used.

#### Parameters

<i>ele_fiber_ID_analysed</i>	(int): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen. -1 will ignore the storage of data for fibers.
<i>fiber_section</i>	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
<i>animate_stress</i>	(bool, optional): Option to show the animation of the fiber stress during the analysis. Defaults to False.
<i>animate_strain</i>	(bool, optional): Option to show the animation of the fiber strain during the analysis. Defaults to False.
<i>fps</i>	(int, optional): Number of frame per seconds for the animations. Defaults to 25.

#### Exceptions

<i>NameError</i>	The methods for the analysis were not called.
------------------	---

Definition at line 670 of file [AnalysisAndPostProcessing.py](#).

```
00670     def FiberResponse(self, ele_fiber_ID_analysed: int, fiber_section = 1, animate_stress = False,
00671                       animate_strain = False, fps = 25):
00672         """
00673         Method that shows the final stress response of the fiber section chosen.
00674         It can also show the animation that shows how the fiber section behaved during the analysis.
00675         The fiber ID and section needs to be recorded during the analysis,
00676         thus if the method LateralForce, Pushover or LoadingProtocol was used, the same fiber ID and
00677         section need to be used.
00678         @param ele_fiber_ID_analysed (int): The ID of the analysed fiber. If fibers are present in the
00679         model and the user wants to save ODB data
00680         (to use in the post-processing with for example FiberResponse), assign to this argument
00681         the ID of the fiber chosen.
00682         -1 will ignore the storage of data for fibers.
00683         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00684         If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00685         @param animate_stress (bool, optional): Option to show the animation of the fiber stress
00686         during the analysis. Defaults to False.
00687         @param animate_strain (bool, optional): Option to show the animation of the fiber strain
00688         during the analysis. Defaults to False.
00689         @param fps (int, optional): Number of frame per seconds for the animations. Defaults to 25.
```

```

00684         @exception NameError: The methods for the analysis were not called.
00685         """
00686         if self.load_case == "None": raise NameError("The analysis is not complete.")
00687
00688         opsplt.plot_fiberResponse2D(self.name_ODB, self.load_case, ele_fiber_ID_analysed,
fiber_section, InputType = 'stress')
00689         if animate_stress:
00690             anil = opsplt.animate_fiberResponse2D(self.name_ODB, self.load_case,
ele_fiber_ID_analysed, fiber_section, InputType = 'stress', fps = fps)
00691         if animate_strain:
00692             anil = opsplt.animate_fiberResponse2D(self.name_ODB, self.load_case,
ele_fiber_ID_analysed, fiber_section, InputType = 'strain', fps = fps)
00693
00694

```

### 7.1.3.3 Gravity()

```

def Gravity (
    self,
    list loaded_nodes,
    list Fy,
    int timeSeries_ID,
    int pattern_ID,
    n_step = 10,
    timeSeries_type = "Linear",
    pattern_type = "Plain",
    constraints_type = "Plain",
    numberer_type = "RCM",
    system_type = "BandGeneral",
    analysis_type = "Static",
    show_plot = False )

```

Method to perform the gravity analysis with vertical loadings (load-control).

It can be used before calling the Pushover or LoadingProtocol methods that perform the actual anlysis. If no vertical loadings present, this method can be avoided.

#### Parameters

<i>loaded_nodes</i>	(list): List of nodes that are loaded by the the forces in Fy. The first node will be recorded (thus usually should be in the roof).
<i>Fy</i>	(list): List of vertical loadings (negative is toward the ground, thus compression; see global coordinate system).
<i>timeSeries_ID</i>	(int): ID of the timeseries.
<i>pattern_ID</i>	(int): ID of the pattern.
<i>n_step</i>	(int, optional): Number of steps used to during the analysis to reach the objective state (with 100% vertical loadings imposed). Defaults to 10.
<i>timeSeries_type</i>	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
<i>pattern_type</i>	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>constraints_type</i>	(str, optional): Type of constraints chosen. It detemines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>numberer_type</i>	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".



## Parameters

<i>system_type</i>	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
<i>analysis_type</i>	(str, optional): Type of analysis chosen. It determines how to construct the <a href="#">Analysis</a> object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
<i>show_plot</i>	(bool, optional): Option to show the 'vertical displacement vs. vertical loading' curve after the analysis. Defaults to False.

## Exceptions

<i>WrongDimension</i>	The dimension of the loaded_nodes and Fy arguments needs to be the same.
<i>NegativeValue</i>	The ID of timeSeries_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of pattern_ID needs to be a positive integer.

Definition at line 60 of file [AnalysisAndPostProcessing.py](#).

```

00061         constraints_type = "Plain", numberer_type = "RCM", system_type = "BandGeneral", analysis_type
= "Static", show_plot = False):
00062     """
00063     Method to perform the gravity analysis with vertical loadings (load-control).
00064     It can be used before calling the Pushover or LoadingProtocol methods that perform the actual
analysis. If no vertical loadings present, this method can be avoided.
00065
00066     @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fy. The first
node will be recorded (thus usually should be in the roof).
00067     @param Fy (list): List of vertical loadings (negative is toward the ground, thus compression;
see global coordinate system).
00068     @param timeSeries_ID (int): ID of the timeseries.
00069     @param pattern_ID (int): ID of the pattern.
00070     @param n_step (int, optional): Number of steps used to during the analysis to reach the
objective state (with 100% vertical loadings imposed). Defaults to 10.
00071     @param timeSeries_type (str, optional): Type of timeseries chosen.
For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00072     @param pattern_type (str, optional): Type of pattern chosen.
For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00073     @param constraints_type (str, optional): Type of constraints chosen. It detemines how the
constraint equations are enforced in the analysis.
For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00074     @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
between equation numbers and degrees-of-freedom.
For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00075     @param system_type (str, optional): Type of system of equations chosen. It determines how to
construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
analysis.
For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
00076     @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
the Analysis object, which defines what type of analysis is to be performed.
For more information, see the OpenSeesPy documentation. Defaults to "Static".
00077     @param show_plot (bool, optional): Option to show the 'vertical displacement vs. vertical
loading' curve after the analysis. Defaults to False.
00078
00079     @exception WrongDimension: The dimension of the loaded_nodes and Fy arguments needs to be the
same.
00080     @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00081     @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00082     """
00083     if len(loaded_nodes) != len(Fy): raise WrongDimension()
00084     if timeSeries_ID < 1: raise NegativeValue()
00085     if pattern_ID < 1: raise NegativeValue()
00086
00087     # for mass defined: opsplt.createODB(self.name_ODB, "Gravity", Nmodes = nEigen);
00088     # for tracking gravity with ODB: opsplt.createODB(self.name_ODB, "Gravity");
00089
00090     # Create load pattern
00091     timeSeries(timeSeries_type, timeSeries_ID)
00092     pattern(pattern_type, timeSeries_ID, pattern_ID)
00093     for ii, node_ID in enumerate(loaded_nodes):
00094         load(node_ID, 0.0, Fy[ii], 0.0) # load(IDNode, Fx, Fy, Mz)
00095         DGravity = 1.0/n_step # load increment
00096
00097
00098
00099
00100
00101
00102

```

```

00103         # Set up analysis options
00104         constraints(constraints_type) # how it handles boundary conditions
00105         numberer(numberer_type)      # renumber dof's to minimize band-width (optimization)
00106         system(system_type)          # how to store and solve the system of equations in the
analysis
00107                                     # For static model, BandGeneral, for transient and/or big
model, UmfPack
00108         integrator("LoadControl", DGravity) # LoadControl and DisplacementControl only with static
model, linear TimeSeries w/ factor of 1
00109                                     # Newmark used for transient model
00110         algorithm("Newton")           # placeholder
00111         analysis(analysis_type)       # define type of analysis: static for pushover
00112
00113         # Analysis
00114         dataG = np.zeros((n_step+1,2))
00115         print("")
00116         print("Gravity analysis starts")
00117         for iteration in range(n_step):
00118             convergence = self.__LoadCtrlLoop(DGravity, iteration,
00119                                               self.algo, self.test_type, self.tol, self.test_opt, self.max_iter)
00120             if convergence != 0: break
00121             dataG[iteration+1,0] = nodeDisp(loaded_nodes[0], 2)/mm_unit
00122             dataG[iteration+1,1] = getLoadFactor(pattern_ID)*Fy[0]/kN_unit
00123
00124         if show_plot:
00125             plt.plot(dataG[:,0], dataG[:,1])
00126             plt.xlabel('Vertical Displacement [mm]')
00127             plt.ylabel('Vertical Load [kN]')
00128             plt.title('Gravity curve')
00129             plt.show()
00130
00131         loadConst("-time", 0.0)
00132
00133         print("")
00134         print("Gravity complete")
00135
00136

```

### 7.1.3.4 LateralForce()

```

def LateralForce (
    self,
    list loaded_nodes,
    list Fx,
    int timeSeries_ID,
    int pattern_ID,
    n_step = 1000,
    fiber_ID_analysed = -1,
    fiber_section = 1,
    timeSeries_type = "Linear",
    pattern_type = "Plain",
    constraints_type = "Plain",
    numberer_type = "RCM",
    system_type = "BandGeneral",
    analysis_type = "Static",
    show_plot = True,
    block = False )

```

Method to perform the lateral force analysis with lateral loading (load-control).

If this method is called, the LoadingProtocol and Pushover methods should be avoided.

#### Parameters

<i>loaded_nodes</i>	(list): List of nodes that are loaded by the the forces in Fx. The first node will be recorded (thus usually should be in the roof).
<i>Fx</i>	(list): List of horizontal loadings (negative is toward left; see global coordinate system).

## Parameters

<i>timeSeries_ID</i>	(int): ID of the timeseries.
<i>pattern_ID</i>	(int): ID of the pattern.
<i>n_step</i>	(int, optional): Number of steps used to during the analysis to reach the objective state (with 100% horizontal loadings imposed). Defaults to 1000.
<i>fiber_ID_analysed</i>	(int, optional): The ID of the analysed fiber. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the fiber chosen. -1 will ignore the storage of data for fibers. Defaults to -1.
<i>fiber_section</i>	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
<i>timeSeries_type</i>	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
<i>pattern_type</i>	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>constraints_type</i>	(str, optional): Type of constraints chosen. It detemines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>numberer_type</i>	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".
<i>system_type</i>	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
<i>analysis_type</i>	(str, optional): Type of analysis chosen. It determines how to construct the <a href="#">Analysis</a> object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
<i>show_plot</i>	(bool, optional): Option to show the 'Horizontal displacement vs. Horizontal loading' curve after the analysis. Defaults to True.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

## Exceptions

<i>WrongDimension</i>	The dimension of the loaded_nodes and Fx arguments needs to be the same.
<i>NegativeValue</i>	The ID of timeSeries_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of pattern_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of fiber_ID_analysed needs to be a positive integer.

Definition at line 137 of file [AnalysisAndPostProcessing.py](#).

```

00139         show_plot = True, block = False):
00140         """
00141         Method to perform the lateral force analysis with lateral loading (load-control).
00142         If this method is called, the LoadingProtocol and Pushover methods should be avoided.
00143
00144         @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fx. The first
00145         node will be recorded (thus usually should be in the roof).
00146         @param Fx (list): List of horizontal loadings (negative is toward left; see global coordinate
00147         system).
00148         @param timeSeries_ID (int): ID of the timeseries.
00149         @param pattern_ID (int): ID of the pattern.
00150         @param n_step (int, optional): Number of steps used to during the analysis to reach the
00151         objective state (with 100% horizontal loadings imposed). Defaults to 1000.
00152         @param fiber_ID_analysed (int, optional): The ID of the analysed fiber. If fibers are present
00153         in the model and the user wants to save ODB data
00154         (to use in the post-processing with for example FiberResponse), assign to this argument
00155         the ID of the fiber chosen.

```

```

00151         -1 will ignore the storage of data for fibers. Defaults to -1.
00152     @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00153         If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00154     @param timeSeries_type (str, optional): Type of timeseries chosen.
00155         For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00156     @param pattern_type (str, optional): Type of pattern chosen.
00157         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00158     @param constraints_type (str, optional): Type of constraints chosen. It detemines how the
00159         constraint equations are enforced in the analysis.
00160         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00161     @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
00162         between equation numbers and degrees-of-freedom.
00163         For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00164     @param system_type (str, optional): Type of system of equations chosen. It determines how to
00165         construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
00166         analysis.
00167         For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
00168     @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
00169         the Analysis object, which defines what type of analysis is to be performed.
00170         For more information, see the OpenSeesPy documentation. Defaults to "Static".
00171     @param show_plot (bool, optional): Option to show the 'Horizontal displacement vs. Horizontal
00172         loading' curve after the analysis. Defaults to True.
00173     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
00174         of the program everytime that a plot should pop up). Defaults to False.
00175
00176     @exception WrongDimension: The dimension of the loaded_nodes and Fx arguments needs to be the
00177         same.
00178     @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00179     @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00180     @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer.
00181
00182     """
00183     if len(loaded_nodes) != len(Fx): raise WrongDimension()
00184     if timeSeries_ID < 1: raise NegativeValue()
00185     if pattern_ID < 1: raise NegativeValue()
00186     if fiber_ID_analysed != -1 and fiber_ID_analysed < 1: raise NegativeValue()
00187
00188     # for mass defined: opsplt.createODB(self.name_ODB, "LateralForce", Nmodes = nEigen);
00189     opsplt.createODB(self.name_ODB, "LateralForce");
00190     if fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODB, "LateralForce",
00191         fiber_ID_analysed, fiber_section)
00192
00193     # Create load pattern
00194     timeSeries(timeSeries_type, timeSeries_ID)
00195     pattern(pattern_type, timeSeries_ID, pattern_ID)
00196     for ii, node_ID in enumerate(loaded_nodes):
00197         load(node_ID, Fx[ii], 0.0, 0.0) # load(IDNode, Fx, Fy, Mz)
00198     force = 1.0/n_step # load increment
00199
00200     # Set up analysis options
00201     constraints(constraints_type) # how it handles boundary conditions
00202     numberer(numberer_type) # renumber dof's to minimize band-width (optimization)
00203     system(system_type) # how to store and solve the system of equations in the
00204
00205     analysis # For static model, BandGeneral, for transient and/or big
00206
00207     model, UmfPack
00208     integrator("LoadControl", force) # LoadControl and DisplacementControl only with static model,
00209     linear TimeSeries w/ factor of 1
00210
00211     # Newmark used for transient model
00212     algorithm("Newton") # placeholder
00213     analysis(analysis_type) # define type of analysis: static for pushover
00214
00215     # Analysis
00216     dataLF = np.zeros((n_step+1,2))
00217     print("")
00218     print("Lateral Force analysis starts")
00219     for iteration in range(n_step):
00220         convergence = self.__LoadCtrlLoop(force, iteration,
00221             self.algo, self.test_type, self.tol, self.test_opt, self.max_iter)
00222         if convergence != 0: break
00223         dataLF[iteration+1,0] = nodeDisp(loaded_nodes[0], 1)/mm_unit
00224         dataLF[iteration+1,1] = getLoadFactor(pattern_ID)*Fx[0]/kN_unit
00225
00226     if show_plot:
00227         plt.plot(dataLF[:,0], dataLF[:,1])
00228         plt.xlabel('Lateral Displacement [mm]')
00229         plt.ylabel('Lateral Load [kN]')
00230         plt.title('Lateral force curve')
00231         if block:
00232             plt.show()
00233
00234     loadConst("-time", 0.0)
00235
00236     print("")
00237     print("Lateral force complete")
00238     self.load_case = "LateralForce"
00239
00240     wipe()

```

00226  
00227

### 7.1.3.5 LoadingProtocol()

```
def LoadingProtocol (
    self,
    int CtrlNode,
    np.ndarray discr_LP,
    int timeSeries_ID,
    int pattern_ID,
    Fx = 1*kN_unit,
    ele_fiber_ID_analysed = -1,
    fiber_section = 1,
    timeSeries_type = "Linear",
    pattern_type = "Plain",
    constraints_type = "Plain",
    numberer_type = "RCM",
    system_type = "UmfPack",
    analysis_type = "Static",
    show_plot = True,
    block = False )
```

Method to perform a loading protocol analysis (displacement-control).

If this method is called, the Pushover and LateralForce methods should be avoided.

#### Parameters

<i>CtrlNode</i>	(int): The node that will be used to impose the displacement from the discr_LP to perform the analysis.
<i>discr_LP</i>	(np.ndarray): The loading protocol array (1 dimension) discretised. It needs to be filled with imposed displacement, not SDR. Use the functions DiscretizeLoadProtocol and DiscretizeLinearly in <a href="#">FunctionalFeatures</a> module to help create and/or discretise one.
<i>timeSeries_ID</i>	(int): ID of the timeseries.
<i>pattern_ID</i>	(int): ID of the pattern.
<i>Fx</i>	(float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarily small. Defaults to 1*kN_unit.
<i>ele_fiber_ID_analysed</i>	(int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen. -1 will ignore the storage of data for fibers. Defaults to -1.
<i>fiber_section</i>	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
<i>timeSeries_type</i>	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
<i>pattern_type</i>	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>constraints_type</i>	(str, optional): Type of constraints chosen. It determines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>numberer_type</i>	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".

## Parameters

<i>system_type</i>	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
<i>analysis_type</i>	(str, optional): Type of analysis chosen. It determines how to construct the <a href="#">Analysis</a> object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
<i>show_plot</i>	(bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

## Exceptions

<i>NegativeValue</i>	The ID of CtrlNode needs to be a positive integer.
<i>NegativeValue</i>	The ID of timeSeries_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of pattern_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of fiber_ID_analysed needs to be a positive integer if is different from -1.

Definition at line 319 of file [AnalysisAndPostProcessing.py](#).

```

00321     show_plot = True, block = False):
00322     """
00323     Method to perform a loading protocol analysis (displacement-control). If this method is
    called, the Pushover and LateralForce methods should be avoided.
00324
00325     @param CtrlNode (int): The node that will be used to impose the displacement from the discr_LP
    to perform the analysis.
00326     @param discr_LP (np.ndarray): The loading protocol array (1 dimension) discretised. It needs
    to be filled with imposed displacement, not SDR.
00327     Use the functions DiscretizeLoadProtocol and DiscretizeLinearly in FunctionalFeatures
    module to help create and/or discretise one.
00328     @param timeSeries_ID (int): ID of the timeseries.
00329     @param pattern_ID (int): ID of the pattern.
00330     @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for
    convergence reasons and it can be arbitrarily small.
00331     Defaults to 1*kN_unit.
00332     @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If
    fibers are present in the model and the user wants to save ODB data
00333     (to use in the post-processing with for example FiberResponse), assign to this argument
    the ID of the element with fibers chosen.
00334     -1 will ignore the storage of data for fibers. Defaults to -1.
00335     @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00336     If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00337     @param timeSeries_type (str, optional): Type of timeseries chosen.
00338     For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00339     @param pattern_type (str, optional): Type of pattern chosen.
00340     For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00341     @param constraints_type (str, optional): Type of constraints chosen. It detemines how the
    constraint equations are enforced in the analysis.
00342     For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00343     @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
    between equation numbers and degrees-of-freedom.
00344     For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00345     @param system_type (str, optional): Type of system of equations chosen. It determines how to
    construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
    analysis.
00346     For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
00347     @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
    the Analysis object, which defines what type of analysis is to be performed.
00348     For more information, see the OpenSeesPy documentation. Defaults to "Static".
00349     @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral
    loading' curve after the analysis. Defaults to True.
00350     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
    of the program everytime that a plot should pop up). Defaults to False.
00351
00352     @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
00353     @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00354     @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00355     @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer if is
    different from -1.

```

```

00356         """
00357         if CtrlNode < 1: raise NegativeValue()
00358         if timeSeries_ID < 1: raise NegativeValue()
00359         if pattern_ID < 1: raise NegativeValue()
00360         if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()
00361
00362         # for mass defined: opsplt.createODB(self.name_ODB, "LoadingProtocol", Nmodes = nEigen);
00363         opsplt.createODB(self.name_ODB, "LoadingProtocol");
00364         if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODB, "LoadingProtocol",
ele_fiber_ID_analysed, fiber_section)
00365
00366         # Create load pattern
00367         timeSeries(timeSeries_type, timeSeries_ID)
00368         pattern(pattern_type, timeSeries_ID, pattern_ID)
00369         load(CtrlNode, Fx, 0.0, 0.0) # load(IDNode, Fx, Fy, Mz)
00370         dU_prev = 0
00371         Nsteps = np.size(discr_LP) # number of pushover analysis steps
00372
00373         # Set up analysis options
00374         constraints(constraints_type) # how it handles boundary conditions
00375         numberer(numberer_type) # renumber dof's to minimize band-width (optimization)
00376         system(system_type) # how to store and solve the system of equations in the
analysis
00377         # For static model, BandGeneral, for transient and/or big
model, UmfPack
00378         integrator("LoadControl", 1) # placeholder
00379         algorithm("Newton") # placeholder
00380         analysis(analysis_type) # define type of analysis: static for LoadingProtocol
00381
00382         # Analysis
00383         dataLP = np.zeros((Nsteps+1,2))
00384         next_step = 0
00385         print("")
00386         print("Loading Protocol analysis starts")
00387         for iteration in range(Nsteps):
00388             # Compute displacement usinf the given loading protocol (discretized)
00389             dU_next = discr_LP[iteration]
00390             dU = dU_next - dU_prev
00391             dU_prev = dU_next
00392
00393             next_step = ProgressingPercentage(Nsteps, iteration, next_step)
00394             convergence = self.__LatDispCtrlLoop(CtrlNode, dU, iteration,
self.algo, self.test_type, self.tol, self.test_opt, self.max_iter,
self.allow_smaller_step)
00396             if convergence != 0: break
00397             dataLP[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
00398             dataLP[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit
00399
00400             if show_plot:
00401                 plt.plot(dataLP[:,0], dataLP[:,1])
00402                 plt.xlabel('Horizontal Displacement [mm]')
00403                 plt.ylabel('Horizontal Load [kN]')
00404                 plt.title('Loading Protocol curve')
00405                 if block:
00406                     plt.show()
00407
00408             print("")
00409             print("Loading Protocol complete")
00410             self.load_case = "LoadingProtocol"
00411
00412             wipe()
00413
00414

```

### 7.1.3.6 Pushover()

```

def Pushover (
    self,
    int CtrlNode,
    Dmax,
    Dincr,
    int timeSeries_ID,
    int pattern_ID,
    Fx = 1*kN_unit,
    ele_fiber_ID_analysed = -1,

```

```

    fiber_section = 1,
    timeSeries_type = "Linear",
    pattern_type = "Plain",
    constraints_type = "Plain",
    numberer_type = "RCM",
    system_type = "UmfPack",
    analysis_type = "Static",
    show_plot = True,
    block = False )

```

Method to perform a pushover analysis (displacement-control).

If this method is called, the LoadingProtocol and LateralForce methods should be avoided.

#### Parameters

<i>CtrlNode</i>	(int): The node that will be used to impose the displacement Dmax of the pushover analysis. If the show_plot option is True, the curve displayed follows this node.
<i>Dmax</i>	(float): The imposed displacement.
<i>Dincr</i>	(float): The incremental displacement to reach Dmax. To converge, it should be small enough (1000 times smaller of Dmax).
<i>timeSeries_ID</i>	(int): ID of the timeseries.
<i>pattern_ID</i>	(int): ID of the pattern.
<i>Fx</i>	(float, optional): The force imposed at the control node CtrlNode. It is used for convergence reasons and it can be arbitrarily small. Defaults to 1*kN_unit.
<i>ele_fiber_ID_analysed</i>	(int, optional): The ID of the analysed element with fibers. If fibers are present in the model and the user wants to save ODB data (to use in the post-processing with for example FiberResponse), assign to this argument the ID of the element with fibers chosen. -1 will ignore the storage of data for fibers. Defaults to -1.
<i>fiber_section</i>	(int, optional): The section number, i.e. the Gauss integratio number. If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
<i>timeSeries_type</i>	(str, optional): Type of timeseries chosen. For more information, see the OpenSeesPy documentation. Defaults to "Linear".
<i>pattern_type</i>	(str, optional): Type of pattern chosen. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>constraints_type</i>	(str, optional): Type of constraints chosen. It determines how the constraint equations are enforced in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "Plain".
<i>numberer_type</i>	(str, optional): Type of numberer chosen. It determines the mapping between equation numbers and degrees-of-freedom. For more information, see the OpenSeesPy documentation. Defaults to "RCM".
<i>system_type</i>	(str, optional): Type of system of equations chosen. It determines how to construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the analysis. For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
<i>analysis_type</i>	(str, optional): Type of analysis chosen. It determines how to construct the <a href="#">Analysis</a> object, which defines what type of analysis is to be performed. For more information, see the OpenSeesPy documentation. Defaults to "Static".
<i>show_plot</i>	(bool, optional): Option to show the 'lateral displacement vs. lateral loading' curve after the analysis. Defaults to True.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.



## Exceptions

<i>NegativeValue</i>	The ID of CtrlNode needs to be a positive integer.
<i>NegativeValue</i>	The ID of timeSeries_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of pattern_ID needs to be a positive integer.
<i>NegativeValue</i>	The ID of ele_fiber_ID_analysed needs to be a positive integer if is different from -1.

Definition at line 228 of file [AnalysisAndPostProcessing.py](#).

```

00230         show_plot = True, block = False):
00231         """
00232         Method to perform a pushover analysis (displacement-control). If this method is called, the
LoadingProtocol and LateralForce methods should be avoided.
00233
00234         @param CtrlNode (int): The node that will be used to impose the displacement Dmax of the
pushover analysis.
00235             If the show_plot option is True, the curve displayed follows this node.
00236         @param Dmax (float): The imposed displacement.
00237         @param Dincr (float): The incremental displacement to reach Dmax. To converge, it should be
small enough (1000 times smaller of Dmax).
00238         @param timeSeries_ID (int): ID of the timeseries.
00239         @param pattern_ID (int): ID of the pattern.
00240         @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for
convergence reasons and it can be arbitrarily small.
00241             Defaults to 1*kN_unit.
00242         @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If
fibers are present in the model and the user wants to save ODB data
00243             (to use in the post-processing with for example FiberResponse), assign to this argument
the ID of the element with fibers chosen.
00244             -1 will ignore the storage of data for fibers. Defaults to -1.
00245         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00246             If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00247         @param timeSeries_type (str, optional): Type of timeseries chosen.
00248             For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00249         @param pattern_type (str, optional): Type of pattern chosen.
00250             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00251         @param constraints_type (str, optional): Type of constraints chosen. It detemines how the
constraint equations are enforced in the analysis.
00252             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00253         @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
between equation numbers and degrees-of-freedom.
00254             For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00255         @param system_type (str, optional): Type of system of equations chosen. It determines how to
construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
analysis.
00256             For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
00257         @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
the Analysis object, which defines what type of analysis is to be performed.
00258             For more information, see the OpenSeesPy documentation. Defaults to "Static".
00259         @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral
loading' curve after the analysis. Defaults to True.
00260         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00261
00262         @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
00263         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00264         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00265         @exception NegativeValue: The ID of ele_fiber_ID_analysed needs to be a positive integer if is
different from -1.
00266         """
00267         if CtrlNode < 1: raise NegativeValue()
00268         if timeSeries_ID < 1: raise NegativeValue()
00269         if pattern_ID < 1: raise NegativeValue()
00270         if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()
00271
00272         # for mass defined: opsplt.createODB(self.name_ODB, "Pushover", Nnodes = nEigen);
00273         opsplt.createODB(self.name_ODB, "Pushover");
00274         if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODB, "Pushover",
ele_fiber_ID_analysed, fiber_section)
00275
00276         # Create load pattern
00277         timeSeries(timeSeries_type, timeSeries_ID)
00278         pattern(pattern_type, timeSeries_ID, pattern_ID)
00279         load(CtrlNode, Fx, 0.0, 0.0) # load(IDNode, Fx, Fy, Mz)
00280         Nsteps = int(abs(Dmax/Dincr)) # number of pushover analysis steps
00281
00282         # Set up analysis options
00283         constraints(constraints_type) # how it handles boundary conditions
00284         numberer(numberer_type) # renumber dof's to minimize band-width (optimization)
00285         system(system_type) # how to store and solve the system of equations in the
analysis

```

```

00286                                     # For static model, BandGeneral, for transient and/or big
00287     model, UmfPack
00287     integrator("LoadControl", 1)      # placeholder
00288     algorithm("Newton")               # placeholder
00289     analysis(analysis_type)           # define type of analysis: static for pushover
00290
00291     # Analysis
00292     dataPO = np.zeros((Nsteps+1,2))
00293     next_step = 0
00294     print("")
00295     print("Pushover analysis starts")
00296     for iteration in range(Nsteps):
00297         next_step = ProgressingPercentage(Nsteps, iteration, next_step)
00298         convergence = self.__LatDispCtrlLoop(CtrlNode, Dincr, iteration,
00299             self.algo, self.test_type, self.tol, self.test_opt, self.max_iter,
self.allow_smaller_step)
00300         if convergence != 0: break
00301         dataPO[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
00302         dataPO[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit
00303
00304     if show_plot:
00305         plt.plot(dataPO[:,0], dataPO[:,1])
00306         plt.xlabel('Horizontal Displacement [mm]')
00307         plt.ylabel('Horizontal Load [kN]')
00308         plt.title('Pushover curve')
00309         if block:
00310             plt.show()
00311
00312     print("")
00313     print("Pushover complete")
00314     self.load_case = "Pushover"
00315
00316     wipe()
00317
00318

```

## 7.1.4 Member Data Documentation

### 7.1.4.1 algo

algo

Definition at line 51 of file [AnalysisAndPostProcessing.py](#).

### 7.1.4.2 allow\_smaller\_step

allow\_smaller\_step

Definition at line 56 of file [AnalysisAndPostProcessing.py](#).

### 7.1.4.3 data\_dir

data\_dir

Definition at line 49 of file [AnalysisAndPostProcessing.py](#).

#### 7.1.4.4 load\_case

load\_case

Definition at line 57 of file [AnalysisAndPostProcessing.py](#).

#### 7.1.4.5 max\_iter

max\_iter

Definition at line 55 of file [AnalysisAndPostProcessing.py](#).

#### 7.1.4.6 name\_ODB

name\_ODB

Definition at line 50 of file [AnalysisAndPostProcessing.py](#).

#### 7.1.4.7 test\_opt

test\_opt

Definition at line 54 of file [AnalysisAndPostProcessing.py](#).

#### 7.1.4.8 test\_type

test\_type

Definition at line 52 of file [AnalysisAndPostProcessing.py](#).

#### 7.1.4.9 tol

tol

Definition at line 53 of file [AnalysisAndPostProcessing.py](#).

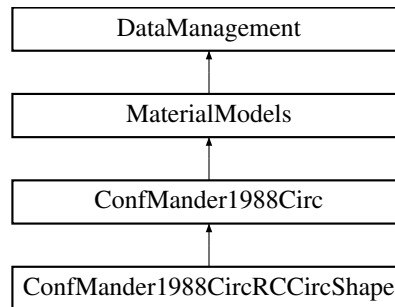
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[AnalysisAndPostProcessing.py](#)

## 7.2 ConfMander1988Circ Class Reference

Class that stores functions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

Inheritance diagram for ConfMander1988Circ:



### Public Member Functions

- def `__init__` (self, int ID, bc, Ac, fc, Ec, nr\_bars, D\_bars, s, D\_hoops, rho\_s\_vol, fs, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)  
*Constructor of the class.*
- def `CheckApplicability` (self)  
*Implementation of the homonym abstract method.*
- def `Compute_ec` (self)  
*Method that computes the compressive concrete yield strain.*
- def `Compute_ecc` (self)  
*Method that computes the compressive confined concrete yield strain.*
- def `Compute_eccu` (self)  
*Method that computes the compressive confined concrete failure strain.*
- def `Compute_ecp` (self)  
*Method that computes the compressive concrete spalling strain.*
- def `Compute_ecu` (self)  
*Method that computes the compressive concrete failure strain.*
- def `Compute_et` (self)  
*Method that computes the tensile concrete yield strain.*
- def `Compute_fct` (self)  
*Method that computes the tensile concrete yield stress.*
- def `Concrete01` (self)  
*Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).*
- def `Concrete04` (self)  
*Generate the material model Concrete04 for circular section confined concrete (Mander 1988).*
- def `Relnit` (self, ec=1, ecp=1, fct=-1, et=-1)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, plot=False, block=False, concrete04=True)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- [Ac](#)
- [Acc](#)
- [Ae](#)
- [bc](#)
- [beta](#)
- [D\\_bars](#)
- [D\\_hoops](#)
- [data](#)
- [Ec](#)
- [ec](#)
- [ecc](#)
- [eccu](#)
- [ecp](#)
- [ecu](#)
- [esu](#)
- [et](#)
- [fc](#)
- [fcc](#)
- [fct](#)
- [fl](#)
- [fl\\_prime](#)
- [fs](#)
- [ID](#)
- [Initialized](#)
- [K\\_combo](#)
- [ke](#)
- [nr\\_bars](#)
- [rho\\_cc](#)
- [rho\\_s\\_vol](#)
- [s](#)
- [section\\_name\\_tag](#)

### 7.2.1 Detailed Description

Class that stores functions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 1994 of file [MaterialModels.py](#).

### 7.2.2 Constructor & Destructor Documentation

### 7.2.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    bc,
    Ac,
    fc,
    Ec,
    nr_bars,
    D_bars,
    s,
    D_hoops,
    rho_s_vol,
    fs,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    esu = -1,
    beta = 0.1 )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>bc</i>	(float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
<i>Ac</i>	(float): Area of the confined core (according to Mander et Al. 1988).
<i>fc</i>	(float): Compressive concrete yield strength (needs to be negative).
<i>Ec</i>	(float): Young modulus.
<i>nr_bars</i>	(float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
<i>D_bars</i>	(float): Diameter of the vertical reinforcing bars.
<i>s</i>	(float): Vertical spacing between hoops.
<i>D_hoops</i>	(float): Diameter of hoops.
<i>rho_s_vol</i>	(float): Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.
<i>fs</i>	(float): Yield stress for the hoops.
<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>esu</i>	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	bc needs to be positive.
<i>NegativeValue</i>	Ac needs to be positive.
<i>PositiveValue</i>	fc needs to be negative.
<i>NegativeValue</i>	Ec needs to be positive.
<i>NegativeValue</i>	nr_bars needs to be positive.
<i>NegativeValue</i>	D_bars needs to be positive.
<i>NegativeValue</i>	s needs to be positive.
<i>NegativeValue</i>	D_hoops needs to be positive.
<i>NegativeValue</i>	rho_s_vol needs to be positive.
<i>NegativeValue</i>	fs needs to be positive.
<i>PositiveValue</i>	ec needs to be negative if different from 1.
<i>PositiveValue</i>	ecp needs to be negative if different from 1.
<i>NegativeValue</i>	fct needs to be positive if different from -1.
<i>NegativeValue</i>	et needs to be positive if different from -1.
<i>NegativeValue</i>	esu needs to be positive if different from -1.

Reimplemented in [ConfMander1988CircRCCircShape](#).

Definition at line 2002 of file [MaterialModels.py](#).

```

02003         ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
02004             """
02005             Constructor of the class.
02006
02007             @param ID (int): Unique material model ID.
02008             @param bc (float): Width of the confined core (from the centerline of the hoops, according to
Mander et Al. 1988).
02009             @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
02010             @param fc (float): Compressive concrete yield strength (needs to be negative).
02011             @param Ec (float): Young modulus.
02012             @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent
nr_bars with different reinforcement areas).
02013             @param D_bars (float): Diameter of the vertical reinforcing bars.
02014             @param s (float): Vertical spacing between hoops.
02015             @param D_hoops (float): Diameter of hoops.
02016             @param rho_s_vol (float): Compute the ratio of the volume of transverse confining steel to the
volume of confined concrete core.
02017             @param fs (float): Yield stress for the hoops.
02018             @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
02019             @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
02020             @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
02021             @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
02022             @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
according to Mander 1988.
02023             @param beta (float, optional): Loading point value defining the exponential curve parameter to
define the residual stress.
02024             Defaults to 0.1 (according to OpenSeesPy documentation)
02025
02026             @exception NegativeValue: ID needs to be a positive integer.
02027             @exception NegativeValue: bc needs to be positive.
02028             @exception NegativeValue: Ac needs to be positive.
02029             @exception PositiveValue: fc needs to be negative.
02030             @exception NegativeValue: Ec needs to be positive.
02031             @exception NegativeValue: nr_bars needs to be positive.
02032             @exception NegativeValue: D_bars needs to be positive.
02033             @exception NegativeValue: s needs to be positive.
02034             @exception NegativeValue: D_hoops needs to be positive.
02035             @exception NegativeValue: rho_s_vol needs to be positive.
02036             @exception NegativeValue: fs needs to be positive.
02037             @exception PositiveValue: ec needs to be negative if different from 1.
02038             @exception PositiveValue: ecp needs to be negative if different from 1.
02039             @exception NegativeValue: fct needs to be positive if different from -1.
02040             @exception NegativeValue: et needs to be positive if different from -1.
02041             @exception NegativeValue: esu needs to be positive if different from -1.

```

```

02042         """
02043         # Check
02044         if ID < 0: raise NegativeValue()
02045         if bc < 0: raise NegativeValue()
02046         if Ac < 0: raise NegativeValue()
02047         if fc > 0: raise PositiveValue()
02048         if Ec < 0: raise NegativeValue()
02049         if nr_bars < 0: raise NegativeValue()
02050         if D_bars < 0: raise NegativeValue()
02051         if s < 0: raise NegativeValue()
02052         if D_hoops < 0: raise NegativeValue()
02053         if rho_s_vol < 0: raise NegativeValue()
02054         if fs < 0: raise NegativeValue()
02055         if ec != 1 and ec > 0: raise PositiveValue()
02056         if ecp != 1 and ecp > 0: raise PositiveValue()
02057         if fct != -1 and fct < 0: raise NegativeValue()
02058         if et != -1 and et < 0: raise NegativeValue()
02059         if esu != -1 and esu < 0: raise NegativeValue()
02060
02061         # Arguments
02062         self.ID = ID
02063         self.bc = bc
02064         self.Ac = Ac
02065         self.fc = fc
02066         self.Ec = Ec
02067         self.nr_bars = nr_bars
02068         self.D_bars = D_bars
02069         self.s = s
02070         self.D_hoops = D_hoops
02071         self.rho_s_vol = rho_s_vol
02072         self.fs = fs
02073         self.esu = 0.05 if esu == -1 else esu
02074         self.beta = beta
02075
02076         # Initialized the parameters that are dependent from others
02077         self.section_name_tag = "None"
02078         self.Initialized = False
02079         self.ReInit(ec, ecp, fct, et)
02080

```

## 7.2.3 Member Function Documentation

### 7.2.3.1 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 2182 of file [MaterialModels.py](#).

```

02182     def CheckApplicability(self):
02183         """
02184         Implementation of the homonym abstract method.
02185         See parent class MaterialModels for detailed information.
02186         """
02187         Check = True
02188         if self.fc < -110*MPa_unit: # Deierlein 1999
02189             Check = False
02190             print("With High Strength concrete (< -110 MPa), a better material model should be used
02191             (see Abdesselam et Al. 2019)")
02192             if not Check:
02193                 print("The validity of the equations is not fullfilled.")
02194                 print("!!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=",
02195                 self.ID)
02196                 print("")

```



### 7.2.3.2 Compute\_ec()

```
def Compute_ec (
    self )
```

Method that computes the compressive concrete yield strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 2197 of file [MaterialModels.py](#).

```
02197     def Compute_ec(self):
02198         """
02199         Method that computes the compressive concrete yield strain.
02200         For more information, see Karthik and Mander 2011.
02201
02202         @returns float: Strain
02203         """
02204         # return -0.002 # Alternative: Mander et Al. 1988
02205         return -0.0015 + self.fc/MPa_unit/70000 # Karthik Mander 2011
02206
02207
```

### 7.2.3.3 Compute\_ecc()

```
def Compute_ecc (
    self )
```

Method that computes the compressive confined concrete yield strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 2249 of file [MaterialModels.py](#).

```
02249     def Compute_ecc(self):
02250         """
02251         Method that computes the compressive confined concrete yield strain.
02252         For more information, see Karthik and Mander 2011.
02253
02254         @returns float: Strain
02255         """
02256         return (1.0 + 5.0 * (self.K_combo-1.0)) * self.ec # Karthik Mander 2011
02257
02258
```

### 7.2.3.4 Compute\_eccu()

```
def Compute_eccu (
    self )
```

Method that computes the compressive confined concrete failure strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 2259 of file [MaterialModels.py](#).

```
02259     def Compute_eccu(self):
02260         """
02261         Method that computes the compressive confined concrete failure strain.
02262         For more information, see Karthik and Mander 2011.
02263
02264         @returns float: Strain
02265         """
02266         # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative:
Prof. Katrin Beyer
02267         return 5*self.ecc # Karthik Mander 2011
02268
02269
```

### 7.2.3.5 Compute\_ecp()

```
def Compute_ecp (
    self )
```

Method that computes the compressive concrete spalling strain.

For more information, see Mander et Al. 1988.

#### Returns

float: Strain

Definition at line 2208 of file [MaterialModels.py](#).

```
02208     def Compute_ecp(self):
02209         """
02210         Method that computes the compressive concrete spalling strain.
02211         For more information, see Mander et Al. 1988.
02212
02213         @returns float: Strain
02214         """
02215         return 2.0*self.ec
02216
02217
```

### 7.2.3.6 Compute\_ecu()

```
def Compute_ecu (
    self )
```

Method that computes the compressive concrete failure strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 2238 of file [MaterialModels.py](#).

```
02238     def Compute_ecu(self):
02239         """
02240         Method that computes the compressive concrete failure strain.
02241         For more information, see Karthik and Mander 2011.
02242
02243         @returns float: Strain
02244         """
02245         # return -0.004 # Alternative: Mander et Al. 1988
02246         return -0.012 - 0.0001 * self.fc/MPa_unit # Karthik Mander 2011
02247
02248
```

### 7.2.3.7 Compute\_et()

```
def Compute_et (
    self )
```

Method that computes the tensile concrete yield strain.

For more information, see Mander et Al. 1988 (eq 45).

#### Returns

float: Strain.

Definition at line 2228 of file [MaterialModels.py](#).

```
02228     def Compute_et(self):
02229         """
02230         Method that computes the tensile concrete yield strain.
02231         For more information, see Mander et Al. 1988 (eq 45).
02232
02233         @returns float: Strain.
02234         """
02235         return self.fct/self.Ec
02236
02237
```

### 7.2.3.8 Compute\_fct()

```
def Compute_fct (
    self )
```

Method that computes the tensile concrete yield stress.

For more information, see SIA 262:2012. Assume that the confinement do not play an essential role in tension.

#### Returns

float: Stress.

Definition at line 2218 of file [MaterialModels.py](#).

```
02218     def Compute_fct(self):
02219         """
02220         Method that computes the tensile concrete yield stress.
02221         For more information, see SIA 262:2012. Assume that the confinement do not play an essential
02222         role in tension.
02223         @returns float: Stress.
02224         """
02225         return 0.30 * math.pow(-self.fc/MPa_unit, 2/3) * MPa_unit
02226
02227
```

### 7.2.3.9 Concrete01()

```
def Concrete01 (
    self )
```

Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).

See `_Concrete01` function for more information. Use this method or `Concrete04`, not both (only one material model for ID).

Definition at line 2270 of file [MaterialModels.py](#).

```
02270     def Concrete01(self):
02271         """
02272         Generate the material model Concrete01 for rectangular section confined concrete (Mander
02273         1988).
02274         See _Concrete01 function for more information. Use this method or Concrete04, not both (only
02275         one material model for ID).
02276         """
02277         _Concrete01(self.ID, self.ecc, self.fcc, self.eccu)
02278         self.Initialized = True
02279         self.UpdateStoredData()
```

### 7.2.3.10 Concrete04()

```
def Concrete04 (
    self )
```

Generate the material model Concrete04 for circular section confined concrete (Mander 1988).

See `_Concrete04` function for more information. Use this method or `Concrete01`, not both (only one material model for ID).

Definition at line 2280 of file [MaterialModels.py](#).

```
02280     def Concrete04(self):
02281         """
02282         Generate the material model Concrete04 for circular section confined concrete (Mander 1988).
02283         See _Concrete04 function for more information. Use this method or Concrete01, not both (only
02284         one material model for ID).
02285         """
02286         _Concrete04(self.ID, self.fcc, self.ecc, self.eccu, self.Ec, self.fct, self.et, self.beta)
02287         self.Initialized = True
02288         self.UpdateStoredData()
02289
```

### 7.2.3.11 ReInit()

```
def ReInit (
    self,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1 )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.

Definition at line 2081 of file [MaterialModels.py](#).

```
02081     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
02082         """
02083         Implementation of the homonym abstract method.
02084         See parent class DataManagement for detailed information.
02085
02086         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
02087         according to Karthik and Mander 2011.
02088         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
02089         to Mander 1988.
02090         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02091         according to SIA 262:2012.
02092         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02093         according to SIA 262:2012.
02094         """
02095         # Check applicability
```

```

02092         self.CheckApplicability()
02093
02094         # Arguments
02095         self.ec = self.Compute_ec() if ec == 1 else ec
02096         self.ecp = self.Compute_ecp() if ecp == 1 else ecp
02097         self.fct = self.Compute_fct() if fct == -1 else fct
02098         self.et = self.Compute_et() if et == -1 else et
02099
02100         # Members
02101         s_prime = self.s - self.D_hoops
02102         self.ecu = self.Compute_ecu()
02103         self.Ae = math.pi/4 * (self.bc - s_prime/2)**2
02104         self.rho_cc = self.nr_bars*self.D_bars**2/4.0*math.pi / self.Ac
02105         self.Acc = self.Ac*(1.0-self.rho_cc)
02106         self.ke = self.Ae/self.Acc
02107         self.fl = -self.rho_s_vol * self.fs / 2
02108         self.fl_prime = self.fl * self.ke
02109         self.K_combo = -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_prime/self.fc) -
2.0*self.fl_prime/self.fc
02110         self.fcc = self.fc * self.K_combo
02111         self.ecc = self.Compute_ecc()
02112         self.eccu = self.Compute_eccu()
02113         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
02114
02115         # Data storage for loading/saving
02116         self.UpdateStoredData()
02117
02118

```

### 7.2.3.12 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False,
    concrete04 = True )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
<i>concrete04</i>	(bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.

Definition at line 2151 of file `MaterialModels.py`.

```

02151     def ShowInfo(self, plot = False, block = False, concrete04 = True):
02152         """
02153         Implementation of the homonym abstract method.
02154         See parent class DataManagement for detailed information.
02155
02156         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
02157         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
02158         program everytime that a plot should pop up). Defaults to False.
02159         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if
False. Defaults to True.
02160         """
02161         print("")
02162         print("Requested info for Confined Mander 1988 (circular) material model Parameters, ID =
{}".format(self.ID))
02163         print("Section associated: {} ".format(self.section_name_tag))
02164         print('Concrete strength fc = {} MPa'.format(self.fc/MPa_unit))
02165         print('Concrete strength confined fcc = {} MPa'.format(self.fcc/MPa_unit))

```

```

02165         print('Strain at maximal strength ec = {}'.format(self.ec))
02166         print('Strain at maximal strength confined ecc = {}'.format(self.ecc))
02167         print('Maximal strain ecu = {}'.format(self.ecu))
02168         print('Maximal strain confined eccu = {}'.format(self.eccu))
02169         print("")
02170
02171         if plot:
02172             fig, ax = plt.subplots()
02173             if concrete04:
02174                 PlotConcrete04(self.fcc, self.Ec, self.ecc, self.eccu, "C", ax, self.ID)
02175             else:
02176                 PlotConcrete01(self.fcc, self.ecc, 0.0, self.eccu, ax, self.ID)
02177
02178             if block:
02179                 plt.show()
02180
02181

```

### 7.2.3.13 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 2120 of file [MaterialModels.py](#).

```

02120     def UpdateStoredData(self):
02121         """
02122         Implementation of the homonym abstract method.
02123         See parent class DataManagement for detailed information.
02124         """
02125         self.data = [{"INFO_TYPE", "ConfMander1988Circ", # Tag for differentiating different data
02126                     ["ID", self.ID],
02127                     ["section_name_tag", self.section_name_tag],
02128                     ["bc", self.bc],
02129                     ["Ac", self.Ac],
02130                     ["fc", self.fc],
02131                     ["Ec", self.Ec],
02132                     ["ec", self.ec],
02133                     ["ecp", self.ecp],
02134                     ["ecu", self.ecu],
02135                     ["fct", self.fct],
02136                     ["et", self.et],
02137                     ["fcc", self.fcc],
02138                     ["ecc", self.ecc],
02139                     ["eccu", self.eccu],
02140                     ["beta", self.beta],
02141                     ["nr_bars", self.nr_bars],
02142                     ["D_bars", self.D_bars],
02143                     ["s", self.s],
02144                     ["D_hoops", self.D_hoops],
02145                     ["rho_s_vol", self.rho_s_vol],
02146                     ["fs", self.fs],
02147                     ["esu", self.esu],
02148                     ["Initialized", self.Initialized]]
02149
02150

```

## 7.2.4 Member Data Documentation

### 7.2.4.1 Ac

Ac

Definition at line 2064 of file [MaterialModels.py](#).

#### 7.2.4.2 Acc

Acc

Definition at line 2105 of file [MaterialModels.py](#).

#### 7.2.4.3 Ae

Ae

Definition at line 2103 of file [MaterialModels.py](#).

#### 7.2.4.4 bc

bc

Definition at line 2063 of file [MaterialModels.py](#).

#### 7.2.4.5 beta

beta

Definition at line 2074 of file [MaterialModels.py](#).

#### 7.2.4.6 D\_bars

D\_bars

Definition at line 2068 of file [MaterialModels.py](#).

#### 7.2.4.7 D\_hoops

D\_hoops

Definition at line 2070 of file [MaterialModels.py](#).



#### 7.2.4.8 data

data

Definition at line 2125 of file [MaterialModels.py](#).

#### 7.2.4.9 Ec

Ec

Definition at line 2066 of file [MaterialModels.py](#).

#### 7.2.4.10 ec

ec

Definition at line 2095 of file [MaterialModels.py](#).

#### 7.2.4.11 ecc

ecc

Definition at line 2111 of file [MaterialModels.py](#).

#### 7.2.4.12 eccu

eccu

Definition at line 2112 of file [MaterialModels.py](#).

#### 7.2.4.13 ecp

ecp

Definition at line 2096 of file [MaterialModels.py](#).

**7.2.4.14 ecu**

ecu

Definition at line [2102](#) of file [MaterialModels.py](#).

**7.2.4.15 esu**

esu

Definition at line [2073](#) of file [MaterialModels.py](#).

**7.2.4.16 et**

et

Definition at line [2098](#) of file [MaterialModels.py](#).

**7.2.4.17 fc**

fc

Definition at line [2065](#) of file [MaterialModels.py](#).

**7.2.4.18 fcc**

fcc

Definition at line [2110](#) of file [MaterialModels.py](#).

**7.2.4.19 fct**

fct

Definition at line [2097](#) of file [MaterialModels.py](#).

#### 7.2.4.20 fl

fl

Definition at line 2107 of file [MaterialModels.py](#).

#### 7.2.4.21 fl\_prime

fl\_prime

Definition at line 2108 of file [MaterialModels.py](#).

#### 7.2.4.22 fs

fs

Definition at line 2072 of file [MaterialModels.py](#).

#### 7.2.4.23 ID

ID

Definition at line 2062 of file [MaterialModels.py](#).

#### 7.2.4.24 Initialized

Initialized

Definition at line 2078 of file [MaterialModels.py](#).

#### 7.2.4.25 K\_combo

K\_combo

Definition at line 2109 of file [MaterialModels.py](#).

#### 7.2.4.26 ke

ke

Definition at line 2106 of file [MaterialModels.py](#).

#### 7.2.4.27 nr\_bars

nr\_bars

Definition at line 2067 of file [MaterialModels.py](#).

#### 7.2.4.28 rho\_cc

rho\_cc

Definition at line 2104 of file [MaterialModels.py](#).

#### 7.2.4.29 rho\_s\_vol

rho\_s\_vol

Definition at line 2071 of file [MaterialModels.py](#).

#### 7.2.4.30 s

s

Definition at line 2069 of file [MaterialModels.py](#).

#### 7.2.4.31 section\_name\_tag

section\_name\_tag

Definition at line 2077 of file [MaterialModels.py](#).

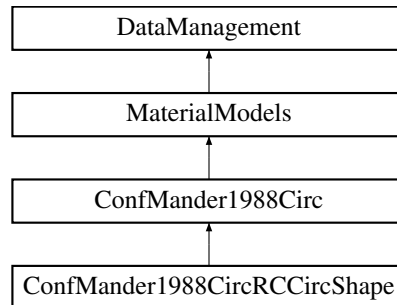
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.3 ConfMander1988CircRCCircShape Class Reference

Class that is the children of [ConfMander1988Circ](#) and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for ConfMander1988CircRCCircShape:



### Public Member Functions

- `def __init__(self, int ID, RCCircShape section, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)`  
*Constructor of the class.*

### Public Attributes

- `section`
- `section_name_tag`

#### 7.3.1 Detailed Description

Class that is the children of [ConfMander1988Circ](#) and combine the class RCCircShape (section) to retrieve the information needed.

#### Parameters

<a href="#">ConfMander1988Circ</a>	Parent class.
------------------------------------	---------------

Definition at line 2290 of file [MaterialModels.py](#).

#### 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RCCircShape section,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    esu = -1,
    beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class `RCCircShape`. The copy of the section passed is stored in the member variable `self.section`.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(RCCircShape): RCCircShape section object.
<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>esu</i>	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from [ConfMander1988Circ](#).

Definition at line 2296 of file [MaterialModels.py](#).

```
02296 def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
02297     """
02298     Constructor of the class. It passes the arguments into the parent class to generate the
02299     combination of the parent class
02300     and the section class RCCircShape.
02301     The copy of the section passed is stored in the member variable self.section.
02302     @param ID (int): Unique material model ID.
02303     @param section (RCCircShape): RCCircShape section object.
02304     @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
02305     according to Karthik and Mander 2011.
02306     @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
02307     to Mander 1988.
02308     @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02309     according to SIA 262:2012.
02310     @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02311     according to SIA 262:2012.
02312     @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
02313     according to Mander 1988.
02314     @param beta (float, optional): Loading point value defining the exponential curve parameter to
02315     define the residual stress.
02316     Defaults to 0.1 (according to OpenSeesPy documentation)
02317     """
02318     self.section = deepcopy(section)
02319     super().__init__(ID, section.bc, section.Ac, section.fc, section.Ec, section.n_bars,
02320                     section.D_bars, section.s, section.D_hoops,
02321                     section.rho_s_vol, section.fs, ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
02322     self.section_name_tag = section.name_tag
```

```
02316         self.UpdateStoredData()
02317
02318
```

### 7.3.3 Member Data Documentation

#### 7.3.3.1 section

section

Definition at line 2312 of file [MaterialModels.py](#).

#### 7.3.3.2 section\_name\_tag

section\_name\_tag

Definition at line 2315 of file [MaterialModels.py](#).

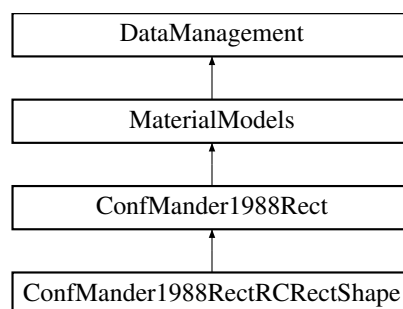
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.4 ConfMander1988Rect Class Reference

Class that stores functions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

Inheritance diagram for ConfMander1988Rect:



## Public Member Functions

- def `__init__` (self, int `ID`, `bc`, `dc`, `Ac`, `fc`, `Ec`, `nr_bars`, `D_bars`, np.ndarray `wx_top`, np.ndarray `wx_bottom`, np.ndarray `wy`, `s`, `D_hoops`, `rho_s_x`, `rho_s_y`, `fs`, `ec`=1, `ecp`=1, `fct`=-1, `et`=-1, `esu`=-1, `beta`=0.1)  
*Constructor of the class.*
- def `CheckApplicability` (self)  
*Implementation of the homonym abstract method.*
- def `Compute_ec` (self)  
*Method that computes the compressive concrete yield strain.*
- def `Compute_ecc` (self)  
*Method that computes the compressive confined concrete yield strain.*
- def `Compute_eccu` (self)  
*Method that computes the compressive confined concrete failure strain.*
- def `Compute_ecp` (self)  
*Method that computes the compressive concrete spalling strain.*
- def `Compute_ecu` (self)  
*Method that computes the compressive concrete failure strain.*
- def `Compute_et` (self)  
*Method that computes the tensile concrete yield strain.*
- def `Compute_fct` (self)  
*Method that computes the tensile concrete yield stress.*
- def `ComputeAi` (self)  
*Method that computes the ineffectual area.*
- def `ComputeConfinementFactor` (self)  
*Method that computes the confinement factor using the digitized table from Mander et Al.*
- def `Concrete01` (self)  
*Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).*
- def `Concrete04` (self)  
*Generate the material model Concrete04 for rectangular section confined concrete (Mander 1988).*
- def `Relnit` (self, `ec`=1, `ecp`=1, `fct`=-1, `et`=-1)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, `plot`=False, `block`=False, `concrete04`=True)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- `Ac`
- `Acc`
- `Ae`
- `Ai`
- `bc`
- `beta`
- `D_bars`
- `D_hoops`
- `data`
- `dc`
- `Ec`
- `ec`
- `ecc`



- [eccu](#)
- [ecp](#)
- [ecu](#)
- [esu](#)
- [et](#)
- [fc](#)
- [fcc](#)
- [fct](#)
- [fl\\_x](#)
- [fl\\_y](#)
- [fs](#)
- [ID](#)
- [Initialized](#)
- [K\\_combo](#)
- [ke](#)
- [nr\\_bars](#)
- [rho\\_cc](#)
- [rho\\_s\\_x](#)
- [rho\\_s\\_y](#)
- [s](#)
- [section\\_name\\_tag](#)
- [wx\\_bottom](#)
- [wx\\_top](#)
- [wy](#)

## Static Public Attributes

- list [array\\_fl2](#) = [None] \* len([curve\\_fl1](#))
- [curve\\_fl1](#) = np.arange(0, 0.3+0.02, 0.02)

### 7.4.1 Detailed Description

Class that stores functions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012. The array [array\\_fl2](#) and curve [curve\\_fl1](#) are the parameter of the digitized table used to extrapolate the confinement factor; they are used as global throughout the [ConfMander1988Rect](#) material model to optimise the program (given the fact that is constant everytime).

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line [1381](#) of file [MaterialModels.py](#).

### 7.4.2 Constructor & Destructor Documentation

### 7.4.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    bc,
    dc,
    Ac,
    fc,
    Ec,
    nr_bars,
    D_bars,
    np.ndarray wx_top,
    np.ndarray wx_bottom,
    np.ndarray wy,
    s,
    D_hoops,
    rho_s_x,
    rho_s_y,
    fs,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    esu = -1,
    beta = 0.1 )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>bc</i>	(float): Width of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
<i>dc</i>	(float): Depth of the confined core (from the centerline of the hoops, according to Mander et Al. 1988).
<i>Ac</i>	(float): Area of the confined core (according to Mander et Al. 1988).
<i>fc</i>	(float): Compressive concrete yield strength (needs to be negative).
<i>Ec</i>	(float): Young modulus.
<i>nr_bars</i>	(float): Number of reinforcement (allow float for computing the equivalent nr_bars with different reinforcement areas).
<i>D_bars</i>	(float): Diameter of the vertical reinforcing bars.
<i>wx_top</i>	(np.ndarray): Vector of 1 dimension that defines the distance between top vertical bars in x direction (NOT CENTERLINE DISTANCES).
<i>wx_bottom</i>	(np.ndarray): Vector of 1 dimension that defines the distance between bottom vertical bars in x direction (NOT CENTERLINE DISTANCES).
<i>wy</i>	(np.ndarray): Vector of 1 dimension that defines the distance between vertical bars in y direction (lateral) (NOT CENTERLINE DISTANCES).
<i>s</i>	(float): Vertical spacing between hoops.
<i>D_hoops</i>	(float): Diameter of hoops.
<i>rho_s_x</i>	(float): Ratio of the transversal area of the hoops to the associated concrete area in the x direction.
<i>rho_s_y</i>	(float): Ratio of the transversal area of the hoops to the associated concrete area in the y direction.
<i>fs</i>	(float): Yield stress for the hoops.

## Parameters

<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>esu</i>	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	bc needs to be positive.
<i>NegativeValue</i>	dc needs to be positive.
<i>NegativeValue</i>	Ac needs to be positive.
<i>PositiveValue</i>	fc needs to be negative.
<i>NegativeValue</i>	Ec needs to be positive.
<i>NegativeValue</i>	nr_bars needs to be positive.
<i>NegativeValue</i>	D_bars needs to be positive.
<i>NegativeValue</i>	s needs to be positive.
<i>NegativeValue</i>	D_hoops needs to be positive.
<i>NegativeValue</i>	rho_s_x needs to be positive.
<i>NegativeValue</i>	rho_s_y needs to be positive.
<i>NegativeValue</i>	fs needs to be positive.
<i>PositiveValue</i>	ec needs to be negative if different from 1.
<i>PositiveValue</i>	ecp needs to be negative if different from 1.
<i>NegativeValue</i>	fct needs to be positive if different from -1.
<i>NegativeValue</i>	et needs to be positive if different from -1.
<i>NegativeValue</i>	esu needs to be positive if different from -1.

Reimplemented in [ConfMander1988RectRCRectShape](#).

Definition at line 1549 of file [MaterialModels.py](#).

```

01550         ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
01551         """
01552         Constructor of the class.
01553
01554         @param ID (int): Unique material model ID.
01555         @param bc (float): Width of the confined core (from the centerline of the hoops, according to
Mander et Al. 1988).
01556         @param dc (float): Depth of the confined core (from the centerline of the hoops, according to
Mander et Al. 1988).
01557         @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
01558         @param fc (float): Compressive concrete yield strength (needs to be negative).
01559         @param Ec (float): Young modulus.
01560         @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent
nr_bars with different reinforcement areas).
01561         @param D_bars (float): Diameter of the vertical reinforcing bars.
01562         @param wx_top (np.ndarray): Vector of 1 dimension that defines the distance between top
vertical bars in x direction (NOT CENTERLINE DISTANCES).
01563         @param wx_bottom (np.ndarray): Vector of 1 dimension that defines the distance between bottom
vertical bars in x direction (NOT CENTERLINE DISTANCES).
01564         @param wy (np.ndarray): Vector of 1 dimension that defines the distance between vertical bars
in y direction (lateral) (NOT CENTERLINE DISTANCES).

```

```

01565         @param s (float): Vertical spacing between hoops.
01566         @param D_hoops (float): Diameter of hoops.
01567         @param rho_s_x (float): Ratio of the transversal area of the hoops to the associated concrete
area in the x direction.
01568         @param rho_s_y (float): Ratio of the transversal area of the hoops to the associated concrete
area in the y direction.
01569         @param fs (float): Yield stress for the hoops.
01570         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
01571         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
01572         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01573         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01574         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
according to Mander 1988.
01575         @param beta (float, optional): Loading point value defining the exponential curve parameter to
define the residual stress.
01576         Defaults to 0.1 (according to OpenSeesPy documentation)
01577
01578         @exception NegativeValue: ID needs to be a positive integer.
01579         @exception NegativeValue: bc needs to be positive.
01580         @exception NegativeValue: dc needs to be positive.
01581         @exception NegativeValue: Ac needs to be positive.
01582         @exception PositiveValue: fc needs to be negative.
01583         @exception NegativeValue: Ec needs to be positive.
01584         @exception NegativeValue: nr_bars needs to be positive.
01585         @exception NegativeValue: D_bars needs to be positive.
01586         @exception NegativeValue: s needs to be positive.
01587         @exception NegativeValue: D_hoops needs to be positive.
01588         @exception NegativeValue: rho_s_x needs to be positive.
01589         @exception NegativeValue: rho_s_y needs to be positive.
01590         @exception NegativeValue: fs needs to be positive.
01591         @exception PositiveValue: ec needs to be negative if different from 1.
01592         @exception PositiveValue: ecp needs to be negative if different from 1.
01593         @exception NegativeValue: fct needs to be positive if different from -1.
01594         @exception NegativeValue: et needs to be positive if different from -1.
01595         @exception NegativeValue: esu needs to be positive if different from -1.
01596         """
01597         # Check
01598         if ID < 1: raise NegativeValue()
01599         if bc < 0: raise NegativeValue()
01600         if dc < 0: raise NegativeValue()
01601         if Ac < 0: raise NegativeValue()
01602         if fc > 0: raise PositiveValue()
01603         if Ec < 0: raise NegativeValue()
01604         if nr_bars < 0: raise NegativeValue()
01605         if D_bars < 0: raise NegativeValue()
01606         if s < 0: raise NegativeValue()
01607         if D_hoops < 0: raise NegativeValue()
01608         if rho_s_x < 0: raise NegativeValue()
01609         if rho_s_y < 0: raise NegativeValue()
01610         if fs < 0: raise NegativeValue()
01611         if ec != 1 and ec > 0: raise PositiveValue()
01612         if ecp != 1 and ecp > 0: raise PositiveValue()
01613         if fct != -1 and fct < 0: raise NegativeValue()
01614         if et != -1 and et < 0: raise NegativeValue()
01615         if esu != -1 and esu < 0: raise NegativeValue()
01616
01617         # Arguments
01618         self.ID = ID
01619         self.bc = bc
01620         self.dc = dc
01621         self.Ac = Ac
01622         self.fc = fc
01623         self.Ec = Ec
01624         self.nr_bars = nr_bars
01625         self.D_bars = D_bars
01626         self.wx_top = copy(wx_top)
01627         self.wx_bottom = copy(wx_bottom)
01628         self.wy = copy(wy)
01629         self.s = s
01630         self.D_hoops = D_hoops
01631         self.rho_s_x = rho_s_x
01632         self.rho_s_y = rho_s_y
01633         self.fs = fs
01634         self.esu = 0.05 if esu == -1 else esu # Mander 1988
01635         self.beta = beta
01636
01637         # Initialized the parameters that are dependent from others
01638         self.section_name_tag = "None"
01639         self.Initialized = False
01640         self.ReInit(ec, ecp, fct, et)
01641

```

### 7.4.3 Member Function Documentation

#### 7.4.3.1 CheckApplicability()

```
def CheckApplicability (
    self )
```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 1756 of file [MaterialModels.py](#).

```
01756     def CheckApplicability(self):
01757         """
01758         Implementation of the homonym abstract method.
01759         See parent class MaterialModels for detailed information.
01760         """
01761         Check = True
01762         if self.fc < -110*MPa_unit: # Deierlein 1999
01763             Check = False
01764             print("With High Strength concrete (< -110 MPa), a better material model should be used
01765             (see Abdesselam et Al. 2019)")
01766             if not Check:
01767                 print("The validity of the equations is not fullfilled.")
01768                 print("!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=",
01769                 self.ID)
01770                 print("")
```

#### 7.4.3.2 Compute\_ec()

```
def Compute_ec (
    self )
```

Method that computes the compressive concrete yield strain.

For more information, see Karthik and Mander 2011.

##### Returns

float: Strain

Definition at line 1771 of file [MaterialModels.py](#).

```
01771     def Compute_ec(self):
01772         """
01773         Method that computes the compressive concrete yield strain.
01774         For more information, see Karthik and Mander 2011.
01775
01776         @returns float: Strain
01777         """
01778         # return -0.002 # Alternative: Mander et Al. 1988
01779         return -0.0015 + self.fc/MPa_unit/70000 # Karthik Mander 2011
01780
01781
```

### 7.4.3.3 Compute\_ecc()

```
def Compute_ecc (
    self )
```

Method that computes the compressive confined concrete yield strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 1823 of file [MaterialModels.py](#).

```
01823     def Compute_ecc(self):
01824         """
01825         Method that computes the compressive confined concrete yield strain.
01826         For more information, see Karthik and Mander 2011.
01827
01828         @returns float: Strain
01829         """
01830         return (1.0 + 5.0 * (self.K_combo-1.0)) * self.ec # Karthik Mander 2011
01831
01832
```

### 7.4.3.4 Compute\_eccu()

```
def Compute_eccu (
    self )
```

Method that computes the compressive confined concrete failure strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 1833 of file [MaterialModels.py](#).

```
01833     def Compute_eccu(self):
01834         """
01835         Method that computes the compressive confined concrete failure strain.
01836         For more information, see Karthik and Mander 2011.
01837
01838         @returns float: Strain
01839         """
01840         # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative:
01841         Prof. Katrin Beyer
01842         return 5*self.ecc # Karthik Mander 2011
01843
```

### 7.4.3.5 Compute\_ecp()

```
def Compute_ecp (
    self )
```

Method that computes the compressive concrete spalling strain.

For more information, see Mander et Al. 1988.

#### Returns

float: Strain

Definition at line 1782 of file [MaterialModels.py](#).

```
01782     def Compute_ecp(self):
01783         """
01784         Method that computes the compressive concrete spalling strain.
01785         For more information, see Mander et Al. 1988.
01786
01787         @returns float: Strain
01788         """
01789         return 2.0*self.ec
01790
01791
```

### 7.4.3.6 Compute\_ecu()

```
def Compute_ecu (
    self )
```

Method that computes the compressive concrete failure strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 1812 of file [MaterialModels.py](#).

```
01812     def Compute_ecu(self):
01813         """
01814         Method that computes the compressive concrete failure strain.
01815         For more information, see Karthik and Mander 2011.
01816
01817         @returns float: Strain
01818         """
01819         # return -0.004 # Alternative: Mander et Al. 1988
01820         return -0.012 - 0.0001 * self.fc/MPa_unit # Karthik Mander 2011
01821
01822
```

#### 7.4.3.7 Compute\_et()

```
def Compute_et (
    self )
```

Method that computes the tensile concrete yield strain.

For more information, see Mander et Al. 1988 (eq 45).

##### Returns

float: Strain.

Definition at line 1802 of file [MaterialModels.py](#).

```
01802     def Compute_et(self):
01803         """
01804         Method that computes the tensile concrete yield strain.
01805         For more information, see Mander et Al. 1988 (eq 45).
01806
01807         @returns float: Strain.
01808         """
01809         return self.fct/self.Ec
01810
01811
```

#### 7.4.3.8 Compute\_fct()

```
def Compute_fct (
    self )
```

Method that computes the tensile concrete yield stress.

For more information, see SIA 262:2012. Assume that the confinement do not play an essential role in tension.

##### Returns

float: Stress.

Definition at line 1792 of file [MaterialModels.py](#).

```
01792     def Compute_fct(self):
01793         """
01794         Method that computes the tensile concrete yield stress.
01795         For more information, see SIA 262:2012. Assume that the confinement do not play an essential
01796         role in tension.
01797
01798         @returns float: Stress.
01799         """
01799         return 0.30 * math.pow(-self.fc/MPa_unit, 2/3) * MPa_unit
01800
01801
```



### 7.4.3.9 ComputeAi()

```
def ComputeAi (
    self )
```

Method that computes the ineffectual area.

For more information, see Mander et Al. 1988.

#### Returns

float: Area.

Definition at line 1844 of file [MaterialModels.py](#).

```
01844     def ComputeAi(self):
01845         """
01846         Method that computes the ineffectual area.
01847         For more information, see Mander et Al. 1988.
01848
01849         @returns float: Area.
01850         """
01851         return ( np.sum(np.multiply(self.wy, self.wy))*2.0 +
01852                 np.sum(np.multiply(self.wx_top, self.wx_top)) +
01853                 np.sum(np.multiply(self.wx_bottom, self.wx_bottom)) ) / 6.0
01854
01855
```

### 7.4.3.10 ComputeConfinementFactor()

```
def ComputeConfinementFactor (
    self )
```

Method that computes the confinement factor using the digitized table from Mander et Al.

1988 that extrapolates the factor using the lateral confining stress in the two direction.

#### Exceptions

<i>NoApplicability</i>	The table from Mander accept ratio of fl/fc smaller than 0.3.
<i>NoApplicability</i>	The table from Mander accept ratio of fl/fc smaller than 0.3.
<i>NegativeValue</i>	fl1_ratio needs to be positive.
<i>NegativeValue</i>	fl2_ratio needs to be positive.

#### Returns

float: Confinement factor.

Definition at line 1856 of file [MaterialModels.py](#).

```
01856     def ComputeConfinementFactor(self):
01857         """
01858         Method that computes the confinement factor using the digitized table from Mander et Al. 1988
01859         that
01860             extrapolates the factor using the lateral confining stress in the two direction.
01861
01862         @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
01862         @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
```

```

01863         @exception NegativeValue: fl1_ratio needs to be positive.
01864         @exception NegativeValue: fl2_ratio needs to be positive.
01865
01866         @returns float: Confinement factor.
01867         """
01868         if self.fl_x == self.fl_y:
01869             return -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_x*self.ke/self.fc) -
01870             2.0*self.fl_x*self.ke/self.fc # in Mander, it has a prime
01871         else:
01872             fl2_ratio = max(self.fl_x*self.ke/self.fc, self.fl_y*self.ke/self.fc)
01873             fl1_ratio = min(self.fl_x*self.ke/self.fc, self.fl_y*self.ke/self.fc)
01874
01875             if fl1_ratio > 0.3: raise NoApplicability()
01876             if fl2_ratio > 0.3: raise NoApplicability()
01877             if fl1_ratio < 0: raise NegativeValue()
01878             if fl2_ratio < 0: raise NegativeValue()
01879
01880             # choose one or two curves
01881             for ii, fl1 in enumerate(curve_fl1):
01882                 if fl1 == fl1_ratio:
01883                     # one curve
01884                     # choose curve
01885                     # curve_fl2 = [curve for ii, curve in enumerate(array_fl2) if index[ii]][0]
01886                     curve_fl2 = array_fl2[ii]
01887
01888                     # Take value (interpolate)
01889                     K = [item[0] for item in curve_fl2]
01890                     fl2 = [item[1] for item in curve_fl2]
01891                     K_res = np.interp(fl2_ratio, fl2, K)
01892
01893                     #TODO: to check fuction:
01894                     # fig, ax = plt.subplots()
01895                     # ax.plot(fl2, K, 'k-')
01896                     # ax.scatter(fl2_ratio, K_res, color='k')
01897                     # ax.grid()
01898                     # plt.show()
01899                     return K_res
01900
01901             # two curves
01902             if fl1 > fl1_ratio:
01903                 fl1_max = fl1
01904                 fl1_min = curve_fl1[ii-1]
01905                 curve_fl2_max = array_fl2[ii]
01906                 curve_fl2_min = array_fl2[ii-1]
01907
01908                 # Take the values (interpolate)
01909                 K_max = [item[0] for item in curve_fl2_max]
01910                 fl2_max = [item[1] for item in curve_fl2_max]
01911                 K_res_max = np.interp(fl2_ratio, fl2_max, K_max)
01912
01913                 K_min = [item[0] for item in curve_fl2_min]
01914                 fl2_min = [item[1] for item in curve_fl2_min]
01915                 K_res_min = np.interp(fl2_ratio, fl2_min, K_min)
01916
01917                 # interpolate with distance from fl1 for fl2
01918                 # should be logarithmic interpolation but error negligible
01919                 K_res = np.interp(fl1_ratio, [fl1_min, fl1_max], [K_res_min, K_res_max])
01920             return K_res
01921

```

### 7.4.3.11 Concrete01()

```

def Concrete01 (
    self )

```

Generate the material model Concrete01 for rectangular section confined concrete (Mander 1988).

See `_Concrete01` function for more information. Use this method or `Concrete04`, not both (only one material model for ID).

Definition at line 1922 of file [MaterialModels.py](#).

```

01922     def Concrete01(self):
01923         """
01924         Generate the material model Concrete01 for rectangular section confined concrete (Mander
1988).

```

```

01925         See _Concrete01 function for more information. Use this method or Concrete04, not both (only
01926         one material model for ID).
01927         """
01928         _Concrete01(self.ID, self.ecc, self.fcc, self.eccu)
01929         self.Initialized = True
01930         self.UpdateStoredData()
01931

```

#### 7.4.3.12 Concrete04()

```

def Concrete04 (
    self )

```

Generate the material model Concrete04 for rectangular section confined concrete (Mander 1988).

See \_Concrete04 function for more information. Use this method or Concrete01, not both (only one material model for ID).

Definition at line 1932 of file [MaterialModels.py](#).

```

01932     def Concrete04(self):
01933         """
01934         Generate the material model Concrete04 for rectangular section confined concrete (Mander
01935         1988). See _Concrete04 function for more information. Use this method or Concrete01, not both (only
01936         one material model for ID).
01937         """
01938         _Concrete04(self.ID, self.fcc, self.ecc, self.eccu, self.Ec, self.fct, self.et, self.beta)
01939         self.Initialized = True
01940         self.UpdateStoredData()
01941

```

#### 7.4.3.13 ReInit()

```

def ReInit (
    self,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1 )

```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.

##### Parameters

<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.

Definition at line 1642 of file [MaterialModels.py](#).

```

01642     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
01643         """
01644         Implementation of the homonym abstract method.
01645         See parent class DataManagement for detailed information.
01646
01647         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01648         according to Karthik and Mander 2011.
01649         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01650         to Mander 1988.
01651         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01652         according to SIA 262:2012.
01653         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01654         according to SIA 262:2012.
01655         """
01656         # Check applicability
01657         self.CheckApplicability()
01658
01659         # Arguments
01660         self.ec = self.Compute_ec() if ec == 1 else ec
01661         self.ecp = self.Compute_ecp() if ecp == 1 else ecp
01662         self.fct = self.Compute_fct() if fct == -1 else fct
01663         self.et = self.Compute_et() if et == -1 else et
01664
01665         # Members (according to Mander 1988, confined concrete)
01666         self.ecu = self.Compute_ecu()
01667         self.Ai = self.ComputeAi()
01668         self.Ae = (self.Ac - self.Ai) * (1.0 - (self.s-self.D_hoops)/2.0/self.bc)*(1.0 -
01669         (self.s-self.D_hoops)/2.0/self.dc)
01670         self.rho_cc = self.nr_bars*self.D_bars**2/4.0*math.pi / self.Ac
01671         self.Acc = self.Ac*(1.0-self.rho_cc)
01672         self.ke = self.Ae/self.Acc
01673         self.fl_x = -self.rho_s_x * self.fs
01674         self.fl_y = -self.rho_s_y * self.fs
01675         self.K_combo = self.ComputeConfinementFactor()
01676         self.fcc = self.fc * self.K_combo
01677         self.ecc = self.Compute_ecc()
01678         self.eccu = self.Compute_eccu()
01679         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
01680         (modified)"
01681
01682         # Data storage for loading/saving
01683         self.UpdateStoredData()
01684
01685
01686
01687
01688
01689

```

#### 7.4.3.14 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False,
    concrete04 = True )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

##### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
<i>concrete04</i>	(bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.

Definition at line 1725 of file `MaterialModels.py`.

```

01725     def ShowInfo(self, plot = False, block = False, concrete04 = True):
01726         """
01727         Implementation of the homonym abstract method.

```

```

01728         See parent class DataManagement for detailed information.
01729
01730         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
01731         False.
01732         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
01733         of the program everytime that a plot should pop up). Defaults to False.
01734         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if
01735         False. Defaults to True.
01736         """
01737         print("")
01738         print("Requested info for Confined Mander 1988 (rectangular) material model Parameters, ID =
01739         {}".format(self.ID))
01740         print("Section associated: {} ".format(self.section_name_tag))
01741         print('Concrete strength fc = {} MPa'.format(self.fc/MPa_unit))
01742         print('Concrete strength confined fcc = {} MPa'.format(self.fcc/MPa_unit))
01743         print('Strain at maximal strength ec = {}'.format(self.ec))
01744         print('Strain at maximal strength confined ecc = {}'.format(self.ecc))
01745         print('Maximal strain ecu = {}'.format(self.ecu))
01746         print('Maximal strain confined eccu = {}'.format(self.eccu))
01747         print("")
01748         if plot:
01749             fig, ax = plt.subplots()
01750             if concrete04:
01751                 PlotConcrete04(self.fcc, self.Ec, self.ecc, self.eccu, "C", ax, self.ID)
01752             else:
01753                 PlotConcrete01(self.fcc, self.ecc, 0.0, self.eccu, ax, self.ID)
01754
01755         if block:
01756             plt.show()
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999

```

#### 7.4.3.15 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 1681 of file `MaterialModels.py`.

```

01681     def UpdateStoredData(self):
01682         """
01683         Implementation of the homonym abstract method.
01684         See parent class DataManagement for detailed information.
01685         """
01686         self.data = [{"INFO_TYPE", "ConfMander1988rect"}, # Tag for differentiating different data
01687                     ["ID", self.ID],
01688                     ["section_name_tag", self.section_name_tag],
01689                     ["bc", self.bc],
01690                     ["dc", self.dc],
01691                     ["Ac", self.Ac],
01692                     ["fc", self.fc],
01693                     ["Ec", self.Ec],
01694                     ["ec", self.ec],
01695                     ["ecp", self.ecp],
01696                     ["ecu", self.ecu],
01697                     ["fct", self.fct],
01698                     ["et", self.et],
01699                     ["fcc", self.fcc],
01700                     ["ecc", self.ecc],
01701                     ["eccu", self.eccu],
01702                     ["beta", self.beta],
01703                     ["nr_bars", self.nr_bars],
01704                     ["D_bars", self.D_bars],
01705                     ["wx_top", self.wx_top],
01706                     ["wx_bottom", self.wx_bottom],
01707                     ["wy", self.wy],
01708                     ["s", self.s],
01709                     ["D_hoops", self.D_hoops],
01710                     ["rho_s_x", self.rho_s_x],
01711                     ["rho_s_y", self.rho_s_y],
01712                     ["fs", self.fs],
01713                     ["esu", self.esu],

```

```
01714         ["Ai", self.Ai],
01715         ["Ae", self.Ae],
01716         ["rho_cc", self.rho_cc],
01717         ["Acc", self.Acc],
01718         ["ke", self.ke],
01719         ["fl_x", self.fl_x],
01720         ["fl_y", self.fl_y],
01721         ["K_combo", self.K_combo],
01722         ["Initialized", self.Initialized]]
01723
01724
```

## 7.4.4 Member Data Documentation

### 7.4.4.1 Ac

Ac

Definition at line 1621 of file [MaterialModels.py](#).

### 7.4.4.2 Acc

Acc

Definition at line 1666 of file [MaterialModels.py](#).

### 7.4.4.3 Ae

Ae

Definition at line 1664 of file [MaterialModels.py](#).

### 7.4.4.4 Ai

Ai

Definition at line 1663 of file [MaterialModels.py](#).

#### 7.4.4.5 array\_fl2

```
list array_fl2 = [None] * len(curve_fl1) [static]
```

Definition at line 1394 of file [MaterialModels.py](#).

#### 7.4.4.6 bc

```
bc
```

Definition at line 1619 of file [MaterialModels.py](#).

#### 7.4.4.7 beta

```
beta
```

Definition at line 1635 of file [MaterialModels.py](#).

#### 7.4.4.8 curve\_fl1

```
curve_fl1 = np.arange(0, 0.3+0.02, 0.02) [static]
```

Definition at line 1393 of file [MaterialModels.py](#).

#### 7.4.4.9 D\_bars

```
D_bars
```

Definition at line 1625 of file [MaterialModels.py](#).

#### 7.4.4.10 D\_hoops

```
D_hoops
```

Definition at line 1630 of file [MaterialModels.py](#).

#### 7.4.4.11 data

data

Definition at line 1686 of file [MaterialModels.py](#).

#### 7.4.4.12 dc

dc

Definition at line 1620 of file [MaterialModels.py](#).

#### 7.4.4.13 Ec

Ec

Definition at line 1623 of file [MaterialModels.py](#).

#### 7.4.4.14 ec

ec

Definition at line 1656 of file [MaterialModels.py](#).

#### 7.4.4.15 ecc

ecc

Definition at line 1672 of file [MaterialModels.py](#).

#### 7.4.4.16 eccu

eccu

Definition at line 1673 of file [MaterialModels.py](#).



#### 7.4.4.17 ecp

ecp

Definition at line 1657 of file [MaterialModels.py](#).

#### 7.4.4.18 ecu

ecu

Definition at line 1662 of file [MaterialModels.py](#).

#### 7.4.4.19 esu

esu

Definition at line 1634 of file [MaterialModels.py](#).

#### 7.4.4.20 et

et

Definition at line 1659 of file [MaterialModels.py](#).

#### 7.4.4.21 fc

fc

Definition at line 1622 of file [MaterialModels.py](#).

#### 7.4.4.22 fcc

fcc

Definition at line 1671 of file [MaterialModels.py](#).

**7.4.4.23 fct**

fct

Definition at line [1658](#) of file [MaterialModels.py](#).

**7.4.4.24 fl\_x**

fl\_x

Definition at line [1668](#) of file [MaterialModels.py](#).

**7.4.4.25 fl\_y**

fl\_y

Definition at line [1669](#) of file [MaterialModels.py](#).

**7.4.4.26 fs**

fs

Definition at line [1633](#) of file [MaterialModels.py](#).

**7.4.4.27 ID**

ID

Definition at line [1618](#) of file [MaterialModels.py](#).

**7.4.4.28 Initialized**

Initialized

Definition at line [1639](#) of file [MaterialModels.py](#).

#### 7.4.4.29 K\_combo

K\_combo

Definition at line 1670 of file [MaterialModels.py](#).

#### 7.4.4.30 ke

ke

Definition at line 1667 of file [MaterialModels.py](#).

#### 7.4.4.31 nrBars

nrBars

Definition at line 1624 of file [MaterialModels.py](#).

#### 7.4.4.32 rho\_cc

rho\_cc

Definition at line 1665 of file [MaterialModels.py](#).

#### 7.4.4.33 rho\_s\_x

rho\_s\_x

Definition at line 1631 of file [MaterialModels.py](#).

#### 7.4.4.34 rho\_s\_y

rho\_s\_y

Definition at line 1632 of file [MaterialModels.py](#).

**7.4.4.35 s**

s

Definition at line 1629 of file [MaterialModels.py](#).

**7.4.4.36 section\_name\_tag**

section\_name\_tag

Definition at line 1638 of file [MaterialModels.py](#).

**7.4.4.37 wx\_bottom**

wx\_bottom

Definition at line 1627 of file [MaterialModels.py](#).

**7.4.4.38 wx\_top**

wx\_top

Definition at line 1626 of file [MaterialModels.py](#).

**7.4.4.39 wy**

wy

Definition at line 1628 of file [MaterialModels.py](#).

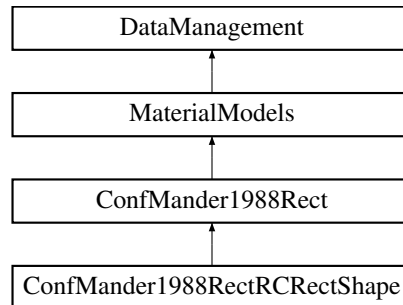
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.5 ConfMander1988RectRCRectShape Class Reference

Class that is the children of [ConfMander1988Rect](#) and combine the class RRectShape (section) to retrieve the information needed.

Inheritance diagram for ConfMander1988RectRCRectShape:



### Public Member Functions

- `def __init__(self, int ID, RRectShape section, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1)`  
*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### Additional Inherited Members

#### 7.5.1 Detailed Description

Class that is the children of [ConfMander1988Rect](#) and combine the class RRectShape (section) to retrieve the information needed.

#### Parameters

<a href="#">ConfMander1988Rect</a>	Parent class.
------------------------------------	---------------

Definition at line 1942 of file [MaterialModels.py](#).

#### 7.5.2 Constructor & Destructor Documentation

### 7.5.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RRectShape section,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    esu = -1,
    beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RRectShape. `wx_bottom`, `wx_top` and `wy` are computed using the private method `__Compute_w` that and the member variable `bars_ranges_position_y` and `bars_position_x` from the section passed. The copy of the section passed is stored in the member variable `self.section`.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(RRectShape): RRectShape section object.
<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>esu</i>	(float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed according to Mander 1988.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from [ConfMander1988Rect](#).

Definition at line 1948 of file [MaterialModels.py](#).

```
01948     def __init__(self, ID: int, section: RRectShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
01949         """
01950         Constructor of the class. It passes the arguments into the parent class to generate the
01951         combination of the parent class
01952         and the section class RRectShape. wx_bottom, wx_top and wy are computed using the private
01953         method __Compute_w that
01954         and the member variable bars_ranges_position_y and bars_position_x from the section
01955         passed.
01956         The copy of the section passed is stored in the member variable self.section.
01957
01958         @param ID (int): Unique material model ID.
01959         @param section (RRectShape): RRectShape section object.
01960         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01961         according to Karthik and Mander 2011.
01962         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01963         to Mander 1988.
01964         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01965         according to SIA 262:2012.
01966         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01967         according to SIA 262:2012.
01968         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
01969         according to Mander 1988.
01970         @param beta (float, optional): Loading point value defining the exponential curve parameter to
01971         define the residual stress.
01972         Defaults to 0.1 (according to OpenSeesPy documentation)
```

```

01964         """
01965         self.section = deepcopy(section)
01966         ranges = section.bars_ranges_position_y
01967         bars = section.bars_position_x
01968         wy = self.__Compute_w(ranges, section.D_bars)
01969         wx_top = self.__Compute_w(bars[0], section.D_bars)
01970         wx_bottom = self.__Compute_w(bars[-1], section.D_bars)
01971
01972         super().__init__(ID, section.bc, section.dc, section.Ac, section.fc, section.Ec,
01973             section.nr_bars, section.D_bars,
01974             wx_top, wx_bottom, wy, section.s, section.D_hoops, section.rho_s_x, section.rho_s_y,
01975             section.fs,
01976             ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
01975         self.section_name_tag = section.name_tag
01976         self.UpdateStoredData()
01977

```

### 7.5.3 Member Data Documentation

#### 7.5.3.1 section

section

Definition at line 1965 of file [MaterialModels.py](#).

#### 7.5.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1975 of file [MaterialModels.py](#).

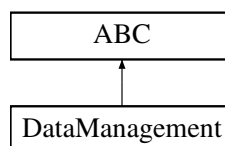
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.6 DataManagement Class Reference

Abstract parent class for data management.

Inheritance diagram for DataManagement:



## Public Member Functions

- def [ReInit](#) (self)  
*Abstract method that computes the value of the parameters with respect of the arguments.*
- def [SaveData](#) (self, f)  
*Function that lists in the command window and saves in a opened file text "f" the data from the "self" class that calls it.*
- def [ShowInfo](#) (self)  
*Abstract method that shows the data stored in the class in the command window.*
- def [UpdateStoredData](#) (self)  
*Abstract method used to define and update the self.data member variable.*

### 7.6.1 Detailed Description

Abstract parent class for data management.

Using the associated MATLAB class

LOAD\_CLASS.m

for the postprocessing in MATLAB, allowing for simpler and more reliable data management because the parameters from the OpenSeesPy analysis are imported automatically.

Definition at line 11 of file [DataManagement.py](#).

### 7.6.2 Member Function Documentation

#### 7.6.2.1 ReInit()

```
def ReInit (
    self )
```

Abstract method that computes the value of the parameters with respect of the arguments.

Use after changing the value of argument inside the class (to update the values accordingly).

This function can be very useful in combination with the function "deepcopy()" from the module "copy".

Be careful that the parameter self.Initialized is also copied, thus it is safer to copy the class before the method that calls the actual OpenSees commands (and initialise the object).

Definition at line 56 of file [DataManagement.py](#).

```
00056     def ReInit(self):
00057         """
00058             Abstract method that computes the value of the parameters with respect of the arguments. \n
00059             Use after changing the value of argument inside the class (to update the values accordingly).
00060             \n
00060             This function can be very useful in combination with the function "deepcopy()" from the module
00061             "copy". \n
00061             Be careful that the parameter self.Initialized is also copied, thus it is safer to copy the
00062             class before the method that calls the actual OpenSees commands (and initialise the object).
00062         """
00063         pass
00064
```



### 7.6.2.2 SaveData()

```
def SaveData (
    self,
    f )
```

Function that lists in the command window and saves in a opened file text "f" the data from the "self" class that calls it.

Example: call this function after this line:  
with open(FileName, 'w') as f:

#### Parameters

<b>f</b>	(io.TextIOWrapper): Opened file to write into
----------	---

#### Exceptions

<b>WrongDimension</b>	The number of lists in the list self.data needs to be 2
-----------------------	---

Definition at line 20 of file [DataManagement.py](#).

```
00020     def SaveData(self, f):
00021         """
00022         Function that lists in the command window and saves in a opened file text "f" the data from
00023         the "self" class that calls it.
00024         Example: call this function after this line: \n
00025         with open(FileName, 'w') as f:
00026
00027             @param f (io.TextIOWrapper): Opened file to write into
00028
00029             @exception WrongDimension: The number of lists in the list self.data needs to be 2
00030             """
00031             if len(self.data[0]) != 2: raise WrongDimension()
00032
00033             delimiter = "#####" # 30 times #
00034             col_delimiter = "\t" # tab
00035             for data_line in self.data:
00036                 f.write('\n')
00037                 for col in data_line:
00038                     if type(col) == np.ndarray:
00039                         tmp_str = np.array_str(col, max_line_width = np.inf)
00040                     else:
00041                         tmp_str = str(col)
00042                     f.write(tmp_str)
00043                     f.write(col_delimiter)
00044                 f.write('\n')
00045                 f.write('NEW INFO SECTION DELIMITER \t')
00046                 f.write(delimiter)
```

### 7.6.2.3 ShowInfo()

```
def ShowInfo (
    self )
```

Abstract method that shows the data stored in the class in the command window.

In some cases, it's possible to plot some information (for example the curve of the material model).

Definition at line 48 of file [DataManagement.py](#).

```
00048     def ShowInfo(self):
```

```

00049         """
00050         Abstract method that shows the data stored in the class in the command window.
00051         In some cases, it's possible to plot some information (for example the curve of the material
model).
00052         """
00053         pass
00054

```

#### 7.6.2.4 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Abstract method used to define and update the self.data member variable.

This member variable (self.data) is a list of lists with 2 entries (info\_name and info\_value) and for each list is stored a different member variable of the class.

Useful to debug the model, export data, copy object.

Definition at line 66 of file [DataManagement.py](#).

```

00066     def UpdateStoredData(self):
00067         """
00068         Abstract method used to define and update the self.data member variable. \n
00069         This member variable (self.data) is a list of lists with 2 entries (info_name and info_value)
00070         and for each list is stored a different member variable of the class. \n
00071         Useful to debug the model, export data, copy object.
00072         """
00073         pass

```

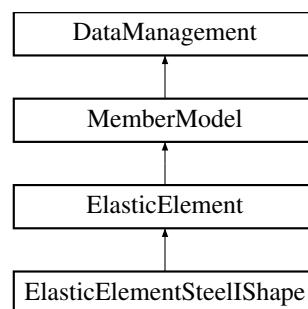
The documentation for this class was generated from the following file:

- [/media/carmin/DATA/Programmi/OpenSeesPyAssistant/DataManagement.py](#)

## 7.7 ElasticElement Class Reference

Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for ElasticElement:



## Public Member Functions

- def `__init__` (self, int iNode\_ID, int jNode\_ID, A, E, ly, int geo\_transf\_ID, ele\_ID=-1)  
*Constructor of the class.*
- def `CreateMember` (self)  
*Method that initialises the member by calling the OpenSeesPy commands through various functions.*
- def `Record` (self, str name\_txt, str data\_dir, force\_rec=True, def\_rec=True, time\_rec=True)  
*Implementation of the homonym abstract method.*
- def `RecordNodeDef` (self, str name\_txt, str data\_dir, time\_rec=True)  
*Implementation of the homonym abstract method.*
- def `Relnit` (self, ele\_ID=-1)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, plot=False, block=False)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- `A`
- `data`
- `E`
- `element_array`
- `element_ID`
- `geo_transf_ID`
- `Initialized`
- `iNode_ID`
- `ly`
- `jNode_ID`
- `section_name_tag`

### 7.7.1 Detailed Description

Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

#### Parameters

<a href="#"><i>MemberModel</i></a>	Parent abstract class.
------------------------------------	------------------------

Definition at line 496 of file [MemberModel.py](#).

### 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    A,
    E,
    Iy,
    int geo_transf_ID,
    ele_ID = -1 )
```

Constructor of the class.

#### Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>A</i>	(float): Area of the member.
<i>E</i>	(float): Young modulus.
<i>Iy</i>	(float): Second moment of inertia (strong axis).
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

#### Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	A needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	Iy needs to be positive.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.

Reimplemented in [ElasticElementSteelShape](#).

Definition at line 502 of file [MemberModel.py](#).

```
00502     def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy, geo_transf_ID: int, ele_ID = -1):
00503         """
00504         Constructor of the class.
00505
00506         @param iNode_ID (int): ID of the first end node.
00507         @param jNode_ID (int): ID of the second end node.
00508         @param A (float): Area of the member.
00509         @param E (float): Young modulus.
00510         @param Iy (float): Second moment of inertia (strong axis).
00511         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00512         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00513
00514         @exception NegativeValue: ID needs to be a positive integer.
00515         @exception NegativeValue: ID needs to be a positive integer.
00516         @exception NegativeValue: A needs to be positive.
00517         @exception NegativeValue: E needs to be positive.
00518         @exception NegativeValue: Iy needs to be positive.
00519         @exception NegativeValue: ID needs to be a positive integer.
00520         @exception NegativeValue: ID needs to be a positive integer.
00521         """
00522         # Check
00523         if iNode_ID < 1: raise NegativeValue()
```

```

00524         if jNode_ID < 1: raise NegativeValue()
00525         if A < 0: raise NegativeValue()
00526         if E < 0: raise NegativeValue()
00527         if Iy < 0: raise NegativeValue()
00528         if geo_transf_ID < 1: raise NegativeValue()
00529         if ele_ID != -1 and ele_ID < 1: raise NegativeValue()
00530
00531         # Arguments
00532         self.iNode_ID = iNode_ID
00533         self.jNode_ID = jNode_ID
00534         self.A = A
00535         self.E = E
00536         self.Iy = Iy
00537         self.geo_transf_ID = geo_transf_ID
00538
00539         # Initialized the parameters that are dependent from others
00540         self.section_name_tag = "None"
00541         self.Initialized = False
00542         self.ReInit(ele_ID = -1)
00543

```

### 7.7.3 Member Function Documentation

#### 7.7.3.1 CreateMember()

```

def CreateMember (
    self )

```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 605 of file [MemberModel.py](#).

```

00605     def CreateMember(self):
00606         """
00607         Method that initialises the member by calling the OpenSeesPy commands through various
00608         functions.
00609         """
00610         self.element_array = [[self.element_ID, self.iNode_ID, self.jNode_ID]]
00611
00612         # Define element
00613         element("elasticBeamColumn", self.element_ID, self.iNode_ID, self.jNode_ID, self.A, self.E,
00614               self.Iy, self.geo_transf_ID)
00615
00616         # Update class
00617         self.Initialized = True
00618         self.UpdateStoredData()

```

#### 7.7.3.2 Record()

```

def Record (
    self,
    str name_txt,
    str data_dir,
    force_rec = True,
    def_rec = True,
    time_rec = True )

```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 619 of file [MemberModel.py](#).

```
00619     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00620         """
00621         Implementation of the homonym abstract method.
00622         See parent class MemberModel for detailed information.
00623         """
00624         super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
00625                        time_rec=time_rec)
00625
00626
```

### 7.7.3.3 RecordNodeDef()

```
def RecordNodeDef (
    self,
    str name_txt,
    str data_dir,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 627 of file [MemberModel.py](#).

```
00627     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00628         """
00629         Implementation of the homonym abstract method.
00630         See parent class MemberModel for detailed information.
00631         """
00632         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
00633
00634
```

### 7.7.3.4 ReInit()

```
def ReInit (
    self,
    ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

#### Parameters

<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use <a href="#">IDConvention</a> to define it.
---------------	--

Definition at line 544 of file [MemberModel.py](#).

```
00544     def ReInit(self, ele_ID = -1):
00545         """
00546         Implementation of the homonym abstract method.
```

```

00547         See parent class DataManagement for detailed information.
00548
00549         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00550         """
00551         # Members
00552         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
00553
00554         # element ID
00555         self.element_ID = IDConvention(self.iNode_ID, self.jNode_ID) if ele_ID == -1 else ele_ID
00556
00557         # Data storage for loading/saving
00558         self.UpdateStoredData()
00559
00560

```

### 7.7.3.5 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 579 of file `MemberModel.py`.

```

00579     def ShowInfo(self, plot = False, block = False):
00580         """
00581         Implementation of the homonym abstract method.
00582         See parent class DataManagement for detailed information.
00583
00584         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00585         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00586         """
00587         print("")
00588         print("Requested info for ElasticElement member model, ID = {}".format(self.element_ID))
00589         print("Section associated {} ".format(self.section_name_tag))
00590         print("Area A = {} mm2".format(self.A/mm2_unit))
00591         print("Young modulus E = {} GPa".format(self.E/GPa_unit))
00592         print("Moment of inertia Iy = {} mm4".format(self.Iy/mm4_unit))
00593         print("Geometric transformation = {}".format(self.geo_transf_ID))
00594         print("")
00595
00596         if plot:
00597             if self.Initialized:
00598                 plot_member(self.element_array, "Elastic Element, ID = {}".format(self.element_ID))
00599                 if block:
00600                     plt.show()
00601             else:
00602                 print("The ElasticElement is not initialized (node and elements not created), ID =
{}".format(self.element_ID))
00603
00604

```

### 7.7.3.6 UpdateStoredData()

```
def UpdateStoredData (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 562 of file [MemberModel.py](#).

```
00562     def UpdateStoredData(self):
00563         """
00564         Implementation of the homonym abstract method.
00565         See parent class DataManagement for detailed information.
00566         """
00567         self.data = [{"INFO_TYPE", "ElasticElement"}, # Tag for differentiating different data
00568                     [{"element_ID", self.element_ID},
00569                      {"section_name_tag", self.section_name_tag},
00570                      {"A", self.A},
00571                      {"E", self.E},
00572                      {"Iy", self.Iy},
00573                      {"iNode_ID", self.iNode_ID},
00574                      {"jNode_ID", self.jNode_ID},
00575                      {"tranf_ID", self.geo_transf_ID},
00576                      {"Initialized", self.Initialized}]]
00577
00578
```

## 7.7.4 Member Data Documentation

### 7.7.4.1 A

A

Definition at line 534 of file [MemberModel.py](#).

### 7.7.4.2 data

data

Definition at line 567 of file [MemberModel.py](#).

### 7.7.4.3 E

E

Definition at line 535 of file [MemberModel.py](#).



#### 7.7.4.4 element\_array

element\_array

Definition at line 609 of file [MemberModel.py](#).

#### 7.7.4.5 element\_ID

element\_ID

Definition at line 555 of file [MemberModel.py](#).

#### 7.7.4.6 geo\_transf\_ID

geo\_transf\_ID

Definition at line 537 of file [MemberModel.py](#).

#### 7.7.4.7 Initialized

Initialized

Definition at line 541 of file [MemberModel.py](#).

#### 7.7.4.8 iNode\_ID

iNode\_ID

Definition at line 532 of file [MemberModel.py](#).

#### 7.7.4.9 Iy

Iy

Definition at line 536 of file [MemberModel.py](#).

#### 7.7.4.10 jNode\_ID

jNode\_ID

Definition at line 533 of file [MemberModel.py](#).

#### 7.7.4.11 section\_name\_tag

section\_name\_tag

Definition at line 540 of file [MemberModel.py](#).

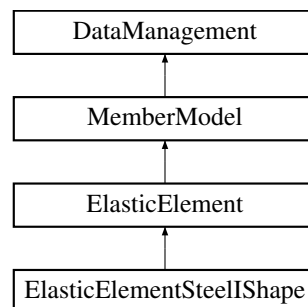
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.8 ElasticElementSteelShape Class Reference

Class that is the children of [ElasticElement](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for ElasticElementSteelShape:



### Public Member Functions

- `def __init__(self, int iNode_ID, int jNode_ID, SteelShape section, int geo_transf_ID, ele_ID=-1)`  
*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.8.1 Detailed Description

Class that is the children of [ElasticElement](#) and combine the class SteelShape (section) to retrieve the information needed.

## Parameters

<a href="#">ElasticElement</a>	Parent class.
--------------------------------	---------------

Definition at line 635 of file [MemberModel.py](#).

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    SteelIShape section,
    int geo_transf_ID,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>section</i>	(SteelIShape): SteelIShape section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [ElasticElement](#).

Definition at line 641 of file [MemberModel.py](#).

```
00641     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int, ele_ID
    = -1):
00642         """
00643         Constructor of the class.
00644
00645         @param iNode_ID (int): ID of the first end node.
00646         @param jNode_ID (int): ID of the second end node.
00647         @param section (SteelIShape): SteelIShape section object.
00648         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00649         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00650         """
00651         self.section = deepcopy(section)
00652         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy, geo_transf_ID, ele_ID)
00653         self.section_name_tag = section.name_tag
00654         self.UpdateStoredData()
00655         # Check length
00656         self._CheckL()
00657
00658
```

## 7.8.3 Member Data Documentation

### 7.8.3.1 section

`section`

Definition at line 651 of file [MemberModel.py](#).

### 7.8.3.2 section\_name\_tag

`section_name_tag`

Definition at line 653 of file [MemberModel.py](#).

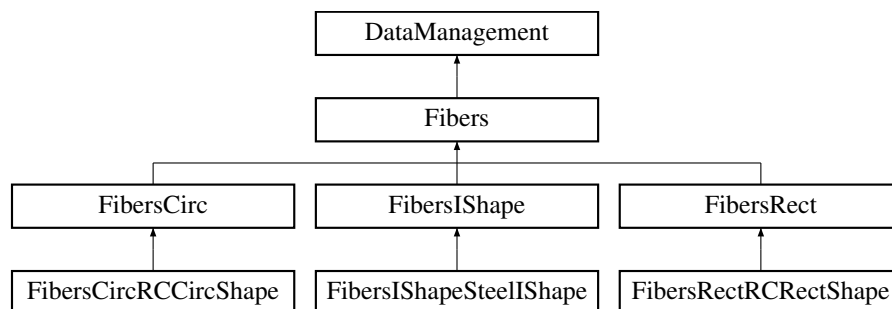
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.9 Fibers Class Reference

Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Inheritance diagram for Fibers:



### 7.9.1 Detailed Description

Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Parameters

<i>DataManagement</i>	Parent abstract class.
-----------------------	------------------------

Definition at line 18 of file [Fibers.py](#).

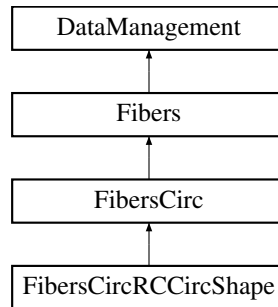
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py](#)

## 7.10 FibersCirc Class Reference

Class that stores functions, material properties, geometric and mechanical parameters for a circular RC fiber section.

Inheritance diagram for FibersCirc:



### Public Member Functions

- `def __init__ (self, int ID, b, e, D_bars, Ay, n_bars, D_hoops, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, list discr_core, list discr_cover, alpha_i=0.0, GJ=0.0)`  
*Constructor of the class.*
- `def CreateFibers (self)`  
*Method that initialise the fiber by calling the OpenSeesPy commands.*
- `def Relnit (self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

### Public Attributes

- `alpha_i`
- `Ay`
- `b`
- `bars_mat_ID`
- `conf_mat_ID`
- `D_bars`
- `D_hoops`
- `data`
- `discr_core`
- `discr_cover`
- `e`
- `fib_sec`
- `GJ`
- `ID`
- `Initialized`
- `n_bars`
- `r_bars`
- `r_core`
- `section_name_tag`
- `unconf_mat_ID`

### 7.10.1 Detailed Description

Class that stores functions, material properties, geometric and mechanical parameters for a circular RC fiber section.

Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

#### Parameters

<a href="#">Fibers</a>	Parent abstract class.
------------------------	------------------------

Definition at line 263 of file [Fibers.py](#).

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    b,
    e,
    D_bars,
    Ay,
    n_bars,
    D_hoops,
    int unconf_mat_ID,
    int conf_mat_ID,
    int bars_mat_ID,
    list discr_core,
    list discr_cover,
    alpha_i = 0.0,
    GJ = 0.0 )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique fiber section ID.
<i>b</i>	(float): Width of the section.
<i>e</i>	(float): Concrete cover.
<i>D_bars</i>	(float): Diameter of vertical reinforcing bars.
<i>Ay</i>	(float): Area of one vertical reinforcing bar.
<i>n_bars</i>	(float): Number of reinforcement (allow float for computing the equivalent n_bars with different reinforcement areas).
<i>D_hoops</i>	(float): Diameter of the hoops.
<i>unconf_mat_ID</i>	(int): ID of material model that will be assigned to the unconfined fibers.
<i>conf_mat_ID</i>	(int): ID of material model that will be assigned to the confined fibers.
<i>bars_mat_ID</i>	(int): ID of material model that will be assigned to the reinforcing bars fibers.

## Parameters

<i>discr_core</i>	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
<i>discr_cover</i>	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
<i>alpha_i</i>	(float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
<i>GJ</i>	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	b needs to be positive.
<i>NegativeValue</i>	e needs to be positive.
<i>InconsistentGeometry</i>	e can't be bigger than half of the width b.
<i>NegativeValue</i>	D_bars needs to be positive.
<i>NegativeValue</i>	Ay needs to be positive.
<i>NegativeValue</i>	n_bars needs to be positive.
<i>NegativeValue</i>	D_hoops needs to be positive.
<i>NegativeValue</i>	unconf_mat_ID needs to be a positive integer.
<i>NegativeValue</i>	conf_mat_ID needs to be a positive integer.
<i>NegativeValue</i>	bars_mat_ID needs to be a positive integer.
<i>WrongDimension</i>	discr_core has a length of 2.
<i>WrongDimension</i>	discr_cover has a length of 2.
<i>NegativeValue</i>	GJ needs to be positive.

Reimplemented in [FibersCircRCCircShape](#).

Definition at line 270 of file [Fibers.py](#).

```

00271     discr_core: list, discr_cover: list, alpha_i = 0.0, GJ = 0.0):
00272     """
00273     Constructor of the class.
00274
00275     @param ID (int): Unique fiber section ID.
00276     @param b (float): Width of the section.
00277     @param e (float): Concrete cover.
00278     @param D_bars (float): Diameter of vertical reinforcing bars.
00279     @param Ay (float): Area of one vertical reinforcing bar.
00280     @param n_bars (float): Number of reinforcement (allow float for computing the equivalent
n_bars with different reinforcement areas).
00281     @param D_hoops (float): Diameter of the hoops.
00282     @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
fibers.
00283     @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00284     @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
fibers.
00285     @param discr_core (list): List with two entries: number of subdivisions (fibers) in the
circumferential direction (number of wedges),
00286     number of subdivisions (fibers) in the radial direction (number of rings) for the confined
core.
00287     @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the
circumferential direction (number of wedges),
00288     number of subdivisions (fibers) in the radial direction (number of rings) for the
unconfined cover.
00289     @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to
the y axis, counterclockwise. Defaults to 0.0.
00290     @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00291

```

```

00292         @exception NegativeValue: ID needs to be a positive integer.
00293         @exception NegativeValue: b needs to be positive.
00294         @exception NegativeValue: e needs to be positive.
00295         @exception InconsistentGeometry: e can't be bigger than half of the width b.
00296         @exception NegativeValue: D_bars needs to be positive.
00297         @exception NegativeValue: Ay needs to be positive.
00298         @exception NegativeValue: n_bars needs to be positive.
00299         @exception NegativeValue: D_hoops needs to be positive.
00300         @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
00301         @exception NegativeValue: conf_mat_ID needs to be a positive integer.
00302         @exception NegativeValue: bars_mat_ID needs to be a positive integer.
00303         @exception WrongDimension: discr_core has a length of 2.
00304         @exception WrongDimension: discr_cover has a length of 2.
00305         @exception NegativeValue: GJ needs to be positive.
00306         """
00307         # Check
00308         if ID < 1: raise NegativeValue()
00309         if b < 0: raise NegativeValue()
00310         if e < 0: raise NegativeValue()
00311         if e > b/2: raise InconsistentGeometry()
00312         if D_bars < 0: raise NegativeValue()
00313         if Ay < 0: raise NegativeValue()
00314         if n_bars < 0: raise NegativeValue()
00315         if D_hoops < 0: raise NegativeValue()
00316         if unconf_mat_ID < 1: raise NegativeValue()
00317         if conf_mat_ID < 1: raise NegativeValue()
00318         if bars_mat_ID < 1: raise NegativeValue()
00319         if len(discr_core) != 2: raise WrongDimension()
00320         if len(discr_cover) != 2: raise WrongDimension()
00321         if GJ < 0: raise NegativeValue()
00322
00323         # Arguments
00324         self.ID = ID
00325         self.b = b
00326         self.e = e
00327         self.D_bars = D_bars
00328         self.Ay = Ay
00329         self.n_bars = n_bars
00330         self.D_hoops = D_hoops
00331         self.unconf_mat_ID = unconf_mat_ID
00332         self.conf_mat_ID = conf_mat_ID
00333         self.bars_mat_ID = bars_mat_ID
00334         self.discr_core = copy(discr_core)
00335         self.discr_cover = copy(discr_cover)
00336         self.alpha_i = alpha_i
00337         self.GJ = GJ
00338
00339         # Initialized the parameters that are dependent from others
00340         self.section_name_tag = "None"
00341         self.Initialized = False
00342         self.ReInit()
00343

```

## 7.10.3 Member Function Documentation

### 7.10.3.1 CreateFibers()

```

def CreateFibers (
    self )

```

Method that initialise the fiber by calling the OpenSeesPy commands.

Definition at line 426 of file [Fibers.py](#).

```

00426     def CreateFibers(self):
00427         """
00428         Method that initialise the fiber by calling the OpenSeesPy commands.
00429         """
00430         create_fiber_section(self.fib_sec)
00431         self.Initialized = True
00432         self.UpdateStoredData()
00433
00434

```



### 7.10.3.2 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 344 of file `Fibers.py`.

```
00344     def ReInit(self):
00345         """
00346         Implementation of the homonym abstract method.
00347         See parent class DataManagement for detailed information.
00348         """
00349         # Memebers
00350         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
00351
00352         # Parameters
00353         self.rBars = self.b/2 - self.e - self.D_hoops - self.D_bars/2
00354         self.r_core = self.b/2 - self.e - self.D_hoops/2
00355
00356         # Create the concrete core fibers
00357         core_cmd = ['patch', 'circ', self.conf_mat_ID, *self.discr_core, 0, 0, 0, self.r_core]
00358
00359         # Create the concrete cover fibers
00360         cover_cmd = ['patch', 'circ', self.unconf_mat_ID, *self.discr_cover, 0, 0, self.r_core,
self.b/2]
00361         self.fib_sec = [['section', 'Fiber', self.ID, '-GJ', self.GJ],
core_cmd, cover_cmd]
00362
00363
00364         # Create the reinforcing fibers
00365         bars_cmd = ['layer', 'circ', self.bars_mat_ID, self.n_bars, self.Ay, 0, 0, self.r_bars,
self.alpha_i]
00366         self.fib_sec.append(bars_cmd)
00367
00368         # Data storage for loading/saving
00369         self.UpdateStoredData()
00370
00371
```

### 7.10.3.3 ShowInfo()

```
def ShowInfo (
    self,
    plot = False,
    block = False )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 399 of file `Fibers.py`.

```
00399     def ShowInfo(self, plot = False, block = False):
00400         """
00401         Implementation of the homonym abstract method.
```

```

00402         See parent class DataManagement for detailed information.
00403
00404         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00405         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00406         """
00407         print("")
00408         print("Requested info for FibersCirc, ID = {}".format(self.ID))
00409         print("Section associated: {} ".format(self.section_name_tag))
00410         print("Base b = {} mm and concrete cover e = {} mm".format(self.b/mm_unit, self.e/mm_unit))
00411         print("Radius of the confined core r_core = {} mm, radius of the bars range rBars = {} mm and
initial angle alpha_i = {} deg".format(self.r_core/mm_unit, self.rBars/mm_unit, self.alpha_i))
00412         print("Confined material model ID = {}".format(self.conf_mat_ID))
00413         print("Unconfined material model ID = {}".format(self.unconf_mat_ID))
00414         print("Bars material model ID = {}".format(self.bars_mat_ID))
00415         print("Discretisation in the core [number of wedges, number of rings] =
{}".format(self.discr_core))
00416         print("Discretisation in the lateral covers [number of wedges, number of rings] =
{}".format(self.discr_cover))
00417         print("")
00418
00419         if plot:
00420             plot_fiber_section(self.fib_sec, matcolor=['#808080', '#D3D3D3', 'k'])
00421
00422         if block:
00423             plt.show()
00424
00425

```

### 7.10.3.4 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 373 of file `Fibers.py`.

```

00373     def UpdateStoredData(self):
00374         """
00375         Implementation of the homonym abstract method.
00376         See parent class DataManagement for detailed information.
00377         """
00378         self.data = [{"INFO_TYPE", "FibersCirc"}, # Tag for differentiating different data
00379                     [{"ID", self.ID},
00380                      ["section_name_tag", self.section_name_tag],
00381                      ["b", self.b],
00382                      ["e", self.e],
00383                      ["r_core", self.r_core],
00384                      ["D_bars", self.D_bars],
00385                      ["Ay", self.Ay],
00386                      ["n_bars", self.n_bars],
00387                      ["r_bars", self.r_bars],
00388                      ["D_hoops", self.D_hoops],
00389                      ["alpha_i", self.alpha_i],
00390                      ["GJ", self.GJ],
00391                      ["conf_mat_ID", self.conf_mat_ID],
00392                      ["discr_core", self.discr_core],
00393                      ["unconf_mat_ID", self.unconf_mat_ID],
00394                      ["discr_cover", self.discr_cover],
00395                      ["bars_mat_ID", self.bars_mat_ID],
00396                      ["Initialized", self.Initialized]]
00397
00398

```

## 7.10.4 Member Data Documentation

#### 7.10.4.1 `alpha_i`

`alpha_i`

Definition at line 336 of file [Fibers.py](#).

#### 7.10.4.2 `Ay`

`Ay`

Definition at line 328 of file [Fibers.py](#).

#### 7.10.4.3 `b`

`b`

Definition at line 325 of file [Fibers.py](#).

#### 7.10.4.4 `bars_mat_ID`

`bars_mat_ID`

Definition at line 333 of file [Fibers.py](#).

#### 7.10.4.5 `conf_mat_ID`

`conf_mat_ID`

Definition at line 332 of file [Fibers.py](#).

#### 7.10.4.6 `D_bars`

`D_bars`

Definition at line 327 of file [Fibers.py](#).

#### 7.10.4.7 D\_hoops

D\_hoops

Definition at line 330 of file [Fibers.py](#).

#### 7.10.4.8 data

data

Definition at line 378 of file [Fibers.py](#).

#### 7.10.4.9 discr\_core

discr\_core

Definition at line 334 of file [Fibers.py](#).

#### 7.10.4.10 discr\_cover

discr\_cover

Definition at line 335 of file [Fibers.py](#).

#### 7.10.4.11 e

e

Definition at line 326 of file [Fibers.py](#).

#### 7.10.4.12 fib\_sec

fib\_sec

Definition at line 361 of file [Fibers.py](#).

#### 7.10.4.13 GJ

GJ

Definition at line 337 of file [Fibers.py](#).

#### 7.10.4.14 ID

ID

Definition at line 324 of file [Fibers.py](#).

#### 7.10.4.15 Initialized

Initialized

Definition at line 341 of file [Fibers.py](#).

#### 7.10.4.16 n\_bars

n\_bars

Definition at line 329 of file [Fibers.py](#).

#### 7.10.4.17 r\_bars

r\_bars

Definition at line 353 of file [Fibers.py](#).

#### 7.10.4.18 r\_core

r\_core

Definition at line 354 of file [Fibers.py](#).

#### 7.10.4.19 section\_name\_tag

section\_name\_tag

Definition at line 340 of file [Fibers.py](#).

#### 7.10.4.20 unconf\_mat\_ID

unconf\_mat\_ID

Definition at line 331 of file [Fibers.py](#).

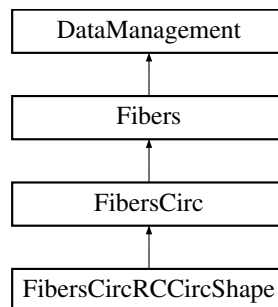
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py](#)

## 7.11 FibersCircRCCircShape Class Reference

Class that is the children of [FibersCirc](#) and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for FibersCircRCCircShape:



### Public Member Functions

- `def __init__(self, int ID, RCCircShape section, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, list discr_core, list discr_cover, alpha_i=0.0, GJ=0)`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.11.1 Detailed Description

Class that is the children of [FibersCirc](#) and combine the class RCCircShape (section) to retrieve the information needed.

## Parameters

<a href="#">FibersCirc</a>	Parent class.
----------------------------	---------------

Definition at line 435 of file [Fibers.py](#).

## 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int ID,
    RCCircShape section,
    int unconf_mat_ID,
    int conf_mat_ID,
    int bars_mat_ID,
    list discr_core,
    list discr_cover,
    alpha_i = 0.0,
    GJ = 0 )
```

Constructor of the class.

## Parameters

<i>ID</i>	(int): Unique fiber section ID.
<i>section</i>	(RCCircShape): RCCircShape section object.
<i>unconf_mat_ID</i>	(int): ID of material model that will be assigned to the unconfined fibers.
<i>conf_mat_ID</i>	(int): ID of material model that will be assigned to the confined fibers.
<i>bars_mat_ID</i>	(int): ID of material model that will be assigned to the reinforcing bars fibers.
<i>discr_core</i>	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the confined core.
<i>discr_cover</i>	(list): List with two entries: number of subdivisions (fibers) in the circumferential direction (number of wedges), number of subdivisions (fibers) in the radial direction (number of rings) for the unconfined cover.
<i>alpha_i</i>	(float, optional): Angle in deg of the first vertical rebars with respect to the y axis, counterclockwise. Defaults to 0.0.
<i>GJ</i>	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Reimplemented from [FibersCirc](#).

Definition at line 441 of file [Fibers.py](#).

```
00442     discr_core: list, discr_cover: list, alpha_i=0.0, GJ=0):
00443     """
00444     Constructor of the class.
00445
00446     @param ID (int): Unique fiber section ID.
```

```

00447         @param section (RCCircShape): RCCircShape section object.
00448         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
fibers.
00449         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00450         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
fibers.
00451         @param discr_core (list): List with two entries: number of subdivisions (fibers) in the
circumferential direction (number of wedges),
00452         number of subdivisions (fibers) in the radial direction (number of rings) for the confined
core.
00453         @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the
circumferential direction (number of wedges),
00454         number of subdivisions (fibers) in the radial direction (number of rings) for the
unconfined cover.
00455         @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to
the y axis, counterclockwise. Defaults to 0.0.
00456         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00457         """
00458         self.section = deepcopy(section)
00459         super().__init__(ID, section.b, section.e, section.D_bars, section.Ay, section.n_bars,
section.D_hoops, unconf_mat_ID, conf_mat_ID, bars_mat_ID,
00460         discr_core, discr_cover, alpha_i=alpha_i, GJ=GJ)
00461         self.section_name_tag = section.name_tag
00462         self.UpdateStoredData()
00463
00464

```

### 7.11.3 Member Data Documentation

#### 7.11.3.1 section

section

Definition at line 458 of file [Fibers.py](#).

#### 7.11.3.2 section\_name\_tag

section\_name\_tag

Definition at line 461 of file [Fibers.py](#).

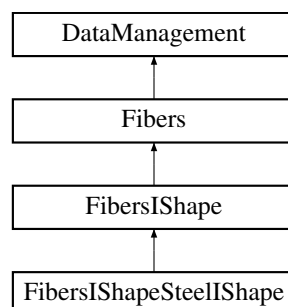
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/Fibers.py](#)

## 7.12 FibersIShape Class Reference

Class that stores functions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.

Inheritance diagram for FibersIShape:





## Public Member Functions

- `def __init__ (self, int ID, d, bf_t, bf_b, tf_t, tf_b, tw, int top_flange_mat_ID, int bottom_flange_mat_ID, int web_mat_ID, list discr_top_flange, list discr_bottom_flange, list discr_web, GJ=0.0)`  
*Constructor of the class.*
- `def CreateFibers (self)`  
*Method that initialise the fiber by calling the OpenSeesPy commands.*
- `def ReInit (self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

## Public Attributes

- `bf_b`
- `bf_t`
- `bottom_flange_mat_ID`
- `d`
- `data`
- `discr_bottom_flange`
- `discr_top_flange`
- `discr_web`
- `fib_sec`
- `GJ`
- `ID`
- `Initialized`
- `section_name_tag`
- `tf_b`
- `tf_t`
- `top_flange_mat_ID`
- `tw`
- `web_mat_ID`

### 7.12.1 Detailed Description

Class that stores functions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.

Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

#### Parameters

<a href="#">Fibers</a>	Parent abstract class.
------------------------	------------------------

Definition at line 465 of file [Fibers.py](#).

## 7.12.2 Constructor & Destructor Documentation

### 7.12.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    d,
    bf_t,
    bf_b,
    tf_t,
    tf_b,
    tw,
    int top_flange_mat_ID,
    int bottom_flange_mat_ID,
    int web_mat_ID,
    list discr_top_flange,
    list discr_bottom_flange,
    list discr_web,
    GJ = 0.0 )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique fiber section ID.
<i>d</i>	(float): Depth of the section.
<i>bf_t</i>	(float): Top flange's width of the section
<i>bf_b</i>	(float): Bottom flange's width of the section
<i>tf_t</i>	(float): Top flange's thickness of the section
<i>tf_b</i>	(float): Bottom flange's thickness of the section
<i>tw</i>	(float): Web's thickness of the section
<i>top_flange_mat_ID</i>	(int): ID of material model that will be assigned to the top flange fibers.
<i>bottom_flange_mat_ID</i>	(int): ID of material model that will be assigned to the bottom flange fibers.
<i>web_mat_ID</i>	(int): ID of material model that will be assigned to the web fibers.
<i>discr_top_flange</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
<i>discr_bottom_flange</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
<i>discr_web</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
<i>GJ</i>	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

#### Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	d needs to be positive.
<i>NegativeValue</i>	bf_t needs to be positive.
<i>NegativeValue</i>	bf_b needs to be positive.
<i>NegativeValue</i>	tf_t needs to be positive.
<i>NegativeValue</i>	tf_b needs to be positive.

## Exceptions

<i>NegativeValue</i>	tw needs to be positive.
<i>NegativeValue</i>	top_flange_mat_ID needs to be a positive integer.
<i>NegativeValue</i>	bottom_flange_mat_ID needs to be a positive integer.
<i>NegativeValue</i>	web_mat_ID needs to be a positive integer.
<i>WrongDimension</i>	discr_top_flange has a length of 2.
<i>WrongDimension</i>	discr_bottom_flange has a length of 2.
<i>WrongDimension</i>	discr_web has a length of 2.
<i>NegativeValue</i>	GJ needs to be positive.
<i>InconsistentGeometry</i>	The sum of the flanges thickness can't be bigger than d.

Reimplemented in [FibersIShapeSteelShape](#).

Definition at line 472 of file [Fibers.py](#).

```

00473     discr_top_flange: list, discr_bottom_flange: list, discr_web: list, GJ = 0.0):
00474     """
00475     Constructor of the class.
00476
00477     @param ID (int): Unique fiber section ID.
00478     @param d (float): Depth of the section.
00479     @param bf_t (float): Top flange's width of the section
00480     @param bf_b (float): Bottom flange's width of the section
00481     @param tf_t (float): Top flange's thickness of the section
00482     @param tf_b (float): Bottom flange's thickness of the section
00483     @param tw (float): Web's thickness of the section
00484     @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange
fibers.
00485     @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom
flange fibers.
00486     @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
00487     @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y)
for the top flange.
00488     @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK
(y) for the bottom flange.
00489     @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
web.
00490     @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00491
00492     @exception NegativeValue: ID needs to be a positive integer.
00493     @exception NegativeValue: d needs to be positive.
00494     @exception NegativeValue: bf_t needs to be positive.
00495     @exception NegativeValue: bf_b needs to be positive.
00496     @exception NegativeValue: tf_t needs to be positive.
00497     @exception NegativeValue: tf_b needs to be positive.
00498     @exception NegativeValue: tw needs to be positive.
00499     @exception NegativeValue: top_flange_mat_ID needs to be a positive integer.
00500     @exception NegativeValue: bottom_flange_mat_ID needs to be a positive integer.
00501     @exception NegativeValue: web_mat_ID needs to be a positive integer.
00502     @exception WrongDimension: discr_top_flange has a length of 2.
00503     @exception WrongDimension: discr_bottom_flange has a length of 2.
00504     @exception WrongDimension: discr_web has a length of 2.
00505     @exception NegativeValue: GJ needs to be positive.
00506     @exception InconsistentGeometry: The sum of the flanges thickness can't be bigger than d.
00507     """
00508     # Check
00509     if ID < 1: raise NegativeValue()
00510     if d < 0: raise NegativeValue()
00511     if bf_t < 0: raise NegativeValue()
00512     if bf_b < 0: raise NegativeValue()
00513     if tf_b < 0: raise NegativeValue()
00514     if tf_t < 0: raise NegativeValue()
00515     if tw < 0: raise NegativeValue()
00516     if top_flange_mat_ID < 1: raise NegativeValue()
00517     if bottom_flange_mat_ID < 1: raise NegativeValue()
00518     if web_mat_ID < 1: raise NegativeValue()
00519     if len(discr_top_flange) != 2: raise WrongDimension()
00520     if len(discr_bottom_flange) != 2: raise WrongDimension()
00521     if len(discr_web) != 2: raise WrongDimension()
00522     if GJ < 0: raise NegativeValue()
00523     if tf_t+tf_b >= d: raise InconsistentGeometry()
00524
00525     # Arguments
00526     self.ID = ID
00527     self.d = d

```

```

00528         self.bf_t = bf_t
00529         self.bf_b = bf_b
00530         self.tf_t = tf_t
00531         self.tf_b = tf_b
00532         self.tw = tw
00533         self.top_flange_mat_ID = top_flange_mat_ID
00534         self.bottom_flange_mat_ID = bottom_flange_mat_ID
00535         self.web_mat_ID = web_mat_ID
00536         self.dscr_top_flange = copy(dscr_top_flange)
00537         self.dscr_bottom_flange = copy(dscr_bottom_flange)
00538         self.dscr_web = copy(dscr_web)
00539         self.GJ = GJ
00540
00541         # Initialized the parameters that are dependent from others
00542         self.section_name_tag = "None"
00543         self.Initialized = False
00544         self.ReInit()
00545

```

## 7.12.3 Member Function Documentation

### 7.12.3.1 CreateFibers()

```

def CreateFibers (
    self )

```

Method that initialise the fiber by calling the OpenSeesPy commands.

Definition at line 631 of file [Fibers.py](#).

```

00631     def CreateFibers(self):
00632         """
00633         Method that initialise the fiber by calling the OpenSeesPy commands.
00634         """
00635         create_fiber_section(self.fib_sec)
00636         self.Initialized = True
00637         self.UpdateStoredData()
00638
00639

```

### 7.12.3.2 ReInit()

```

def ReInit (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 546 of file [Fibers.py](#).

```

00546     def ReInit(self):
00547         """
00548         Implementation of the homonym abstract method.
00549         See parent class DataManagement for detailed information.
00550         """
00551         # Memebers
00552         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
00553
00554         # Parameters
00555         z1 = self.tw/2
00556         y1 = (self.d - self.tf_t - self.tf_b)/2
00557

```

```

00558         # Create the flange top
00559         flange_top = [y1, -self.bf_t/2, y1+self.tf_t, self.bf_t/2]
00560         flange_top_cmd = ['patch', 'rect', self.top_flange_mat_ID, *self.dscr_top_flange,
*flange_top]
00561
00562         # Create the flange bottom
00563         flange_bottom = [-y1-self.tf_b, -self.bf_b/2, -y1, self.bf_b/2]
00564         flange_bottom_cmd = ['patch', 'rect', self.bottom_flange_mat_ID, *self.dscr_bottom_flange,
*flange_bottom]
00565
00566         # Create the web
00567         web = [-y1, -z1, y1, z1]
00568         web_cmd = ['patch', 'rect', self.web_mat_ID, *self.dscr_web, *web]
00569
00570         self.fib_sec = [['section', 'Fiber', self.ID, '-GJ', self.GJ],
            flange_top_cmd, web_cmd, flange_bottom_cmd]
00571
00572
00573         # Data storage for loading/saving
00574         self.UpdateStoredData()
00575
00576

```

### 7.12.3.3 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the fiber. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 602 of file [Fibers.py](#).

```

00602     def ShowInfo(self, plot = False, block = False):
00603         """
00604         Implementation of the homonym abstract method.
00605         See parent class DataManagement for detailed information.
00606
00607         @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
00608         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00609         """
00610         print("")
00611         print("Requested info for FibersRect, ID = {}".format(self.ID))
00612         print("Section associated: {} ".format(self.section_name_tag))
00613         print("Depth d = {} mm and web thickness tw = {} mm".format(self.d/mm_unit, self.tw/mm_unit))
00614         print("Top flange width bf_t = {} mm and thickness tf_t = {} mm".format(self.bf_t/mm_unit,
self.tf_t/mm_unit))
00615         print("Bottom flange width bf_b = {} mm and thickness tf_b = {} mm".format(self.bf_b/mm_unit,
self.tf_b/mm_unit))
00616         print("Web material model ID = {}".format(self.web_mat_ID))
00617         print("Top flange material model ID = {}".format(self.top_flange_mat_ID))
00618         print("Bottom flange material model ID = {}".format(self.bottom_flange_mat_ID))
00619         print("Discretisation in the web [IJ or x/z dir, JK or y dir] = {}".format(self.dscr_web))
00620         print("Discretisation in the top flange [IJ or x/z dir, JK or y dir] =
{}".format(self.dscr_top_flange))
00621         print("Discretisation in the bottom flange [IJ or x/z dir, JK or y dir] =
{}".format(self.dscr_bottom_flange))
00622         print("")
00623
00624         if plot:
00625             plot\_fiber\_section(self.fib_sec, matcolor=['r', 'b', 'g', 'k'])

```

```

00626
00627         if block:
00628             plt.show()
00629
00630

```

#### 7.12.3.4 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 578 of file [Fibers.py](#).

```

00578     def UpdateStoredData(self):
00579         """
00580         Implementation of the homonym abstract method.
00581         See parent class DataManagement for detailed information.
00582         """
00583         self.data = [{"INFO_TYPE", "FibersIShape"}, # Tag for differentiating different data
00584                     ["ID", self.ID],
00585                     ["section_name_tag", self.section_name_tag],
00586                     ["d", self.d],
00587                     ["bf_t", self.bf_t],
00588                     ["bf_b", self.bf_b],
00589                     ["tf_t", self.tf_t],
00590                     ["tf_b", self.tf_b],
00591                     ["tw", self.tw],
00592                     ["GJ", self.GJ],
00593                     ["top_flange_mat_ID", self.top_flange_mat_ID],
00594                     ["bottom_flange_mat_ID", self.bottom_flange_mat_ID],
00595                     ["web_mat_ID", self.web_mat_ID],
00596                     ["discr_top_flange", self.discr_top_flange],
00597                     ["discr_bottom_flange", self.discr_bottom_flange],
00598                     ["discr_web", self.discr_web],
00599                     ["Initialized", self.Initialized]]
00600
00601

```

### 7.12.4 Member Data Documentation

#### 7.12.4.1 bf\_b

`bf_b`

Definition at line 529 of file [Fibers.py](#).

#### 7.12.4.2 bf\_t

`bf_t`

Definition at line 528 of file [Fibers.py](#).

#### 7.12.4.3 bottom\_flange\_mat\_ID

bottom\_flange\_mat\_ID

Definition at line 534 of file [Fibers.py](#).

#### 7.12.4.4 d

d

Definition at line 527 of file [Fibers.py](#).

#### 7.12.4.5 data

data

Definition at line 583 of file [Fibers.py](#).

#### 7.12.4.6 discr\_bottom\_flange

discr\_bottom\_flange

Definition at line 537 of file [Fibers.py](#).

#### 7.12.4.7 discr\_top\_flange

discr\_top\_flange

Definition at line 536 of file [Fibers.py](#).

#### 7.12.4.8 discr\_web

discr\_web

Definition at line 538 of file [Fibers.py](#).

**7.12.4.9 fib\_sec**

`fib_sec`

Definition at line 570 of file [Fibers.py](#).

**7.12.4.10 GJ**

`GJ`

Definition at line 539 of file [Fibers.py](#).

**7.12.4.11 ID**

`ID`

Definition at line 526 of file [Fibers.py](#).

**7.12.4.12 Initialized**

`Initialized`

Definition at line 543 of file [Fibers.py](#).

**7.12.4.13 section\_name\_tag**

`section_name_tag`

Definition at line 542 of file [Fibers.py](#).

**7.12.4.14 tf\_b**

`tf_b`

Definition at line 531 of file [Fibers.py](#).



#### 7.12.4.15 `tf_t`

`tf_t`

Definition at line 530 of file [Fibers.py](#).

#### 7.12.4.16 `top_flange_mat_ID`

`top_flange_mat_ID`

Definition at line 533 of file [Fibers.py](#).

#### 7.12.4.17 `tw`

`tw`

Definition at line 532 of file [Fibers.py](#).

#### 7.12.4.18 `web_mat_ID`

`web_mat_ID`

Definition at line 535 of file [Fibers.py](#).

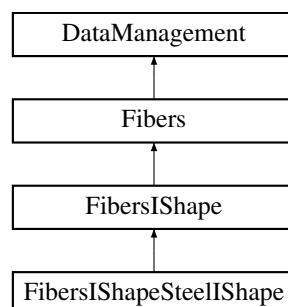
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py](#)

## 7.13 FibersIShapeSteelShape Class Reference

Class that is the children of [FibersIShape](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

Inheritance diagram for [FibersIShapeSteelShape](#):



## Public Member Functions

- `def __init__ (self, int ID, SteelShape section, int top\_flange\_mat\_ID, list discr\_top\_flange, list discr\_bottom\_flange, list discr\_web, GJ=0.0, bottom\_flange\_mat\_ID=-1, web\_mat\_ID=-1)`

*Constructor of the class.*

## Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.13.1 Detailed Description

Class that is the children of [FibersIShape](#) and combine the class [SteelShape](#) ([section](#)) to retrieve the information needed.

#### Parameters

<a href="#">FibersIShape</a>	Parent class.
------------------------------	---------------

Definition at line [640](#) of file [Fibers.py](#).

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    SteelShape section,
    int top_flange_mat_ID,
    list discr_top_flange,
    list discr_bottom_flange,
    list discr_web,
    GJ = 0.0,
    bottom_flange_mat_ID = -1,
    web_mat_ID = -1 )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique fiber section ID.
<i>section</i>	(SteelShape): SteelShape section object.

## Parameters

<i>top_flange_mat_ID</i>	(int): ID of material model that will be assigned to the top flange fibers.
<i>discr_top_flange</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top flange.
<i>discr_bottom_flange</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the bottom flange.
<i>discr_web</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the web.
<i>GJ</i>	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.
<i>bottom_flange_mat_ID</i>	(int): ID of material model that will be assigned to the bottom flange fibers. Defaults to -1, e.g. equal to top_flange_mat_ID.
<i>web_mat_ID</i>	(int): ID of material model that will be assigned to the web fibers. Defaults to -1, e.g. equal to top_flange_mat_ID.

Reimplemented from [FibersIShape](#).

Definition at line 646 of file [Fibers.py](#).

```

00647         GJ=0.0, bottom_flange_mat_ID = -1, web_mat_ID = -1):
00648             """
00649             Constructor of the class.
00650
00651             @param ID (int): Unique fiber section ID.
00652             @param section (SteelIShape): SteelIShape section object.
00653             @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange
fibers.
00654             @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y)
for the top flange.
00655             @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK
(y) for the bottom flange.
00656             @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
web.
00657             @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00658             @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom
flange fibers.
00659                 Defaults to -1, e.g. equal to top_flange_mat_ID.
00660             @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
00661                 Defaults to -1, e.g. equal to top_flange_mat_ID.
00662             """
00663             self.section = deepcopy(section)
00664             if bottom_flange_mat_ID == -1: bottom_flange_mat_ID = top_flange_mat_ID
00665             if web_mat_ID == -1: web_mat_ID = top_flange_mat_ID
00666
00667             super().__init__(ID, section.d, section.bf, section.bf, section.tf, section.tf, section.tw,
top_flange_mat_ID, bottom_flange_mat_ID, web_mat_ID,
00668                 discr_top_flange, discr_bottom_flange, discr_web, GJ)
00669             self.section_name_tag = section.name_tag
00670             self.UpdateStoredData()
00671
00672

```

## 7.13.3 Member Data Documentation

### 7.13.3.1 section

section

Definition at line 663 of file [Fibers.py](#).

### 7.13.3.2 section\_name\_tag

section\_name\_tag

Definition at line 669 of file [Fibers.py](#).

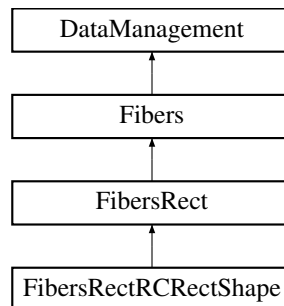
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/Fibers.py](#)

## 7.14 FibersRect Class Reference

Class that stores functions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.

Inheritance diagram for FibersRect:



### Public Member Functions

- `def __init__ (self, int ID, b, d, Ay, D_hoops, e, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, np.ndarray bars_x, np.ndarray ranges_y, list discr_core, list discr_cover_lateral, list discr_cover_topbottom, GJ=0.0)`  
*Constructor of the class.*
- `def CreateFibers (self)`  
*Method that initialises the fiber by calling the OpenSeesPy commands.*
- `def Relnit (self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

## Public Attributes

- [Ay](#)
- [b](#)
- [bars\\_mat\\_ID](#)
- [bars\\_x](#)
- [conf\\_mat\\_ID](#)
- [d](#)
- [D\\_hoops](#)
- [data](#)
- [discr\\_core](#)
- [discr\\_cover\\_lateral](#)
- [discr\\_cover\\_topbottom](#)
- [e](#)
- [fib\\_sec](#)
- [GJ](#)
- [ID](#)
- [Initialized](#)
- [ranges\\_y](#)
- [rebarYZ](#)
- [section\\_name\\_tag](#)
- [unconf\\_mat\\_ID](#)

### 7.14.1 Detailed Description

Class that stores functions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.

Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more information, see the OpenSeesPy documentation.

#### Parameters

<a href="#">Fibers</a>	Parent abstract class.
------------------------	------------------------

Definition at line 28 of file [Fibers.py](#).

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    b,
    d,
    Ay,
```

```

        D_hoops,
        e,
        int unconf_mat_ID,
        int conf_mat_ID,
        int bars_mat_ID,
        np.ndarray bars_x,
        np.ndarray ranges_y,
        list discr_core,
        list discr_cover_lateral,
        list discr_cover_topbottom,
        GJ = 0.0 )

```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique fiber section ID.
<i>b</i>	(float): Width of the section.
<i>d</i>	(float): Depth of the section.
<i>Ay</i>	(float): Area of one vertical reinforcing bar.
<i>D_hoops</i>	(float): Diameter of the hoops.
<i>e</i>	(float): Concrete cover.
<i>unconf_mat_ID</i>	(int): ID of material model that will be assigned to the unconfined fibers.
<i>conf_mat_ID</i>	(int): ID of material model that will be assigned to the confined fibers.
<i>bars_mat_ID</i>	(int): ID of material model that will be assigned to the reinforcing bars fibers.
<i>bars_x</i>	(np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction. Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned). Starting from the left to right, from the top range to the bottom one. The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object"
<i>ranges_y</i>	(np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_x. Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction. Starting from the top range to the bottom one.
<i>discr_core</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
<i>discr_cover_lateral</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined cover.
<i>discr_cover_topbottom</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined cover.
<i>GJ</i>	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

#### Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	b needs to be positive.
<i>NegativeValue</i>	d needs to be positive.
<i>NegativeValue</i>	Ay needs to be positive.
<i>NegativeValue</i>	D_hoops needs to be positive.
<i>NegativeValue</i>	e needs to be positive.
<i>NegativeValue</i>	unconf_mat_ID needs to be a positive integer.
<i>NegativeValue</i>	conf_mat_ID needs to be a positive integer.
<i>NegativeValue</i>	bars_mat_ID needs to be a positive integer.

## Exceptions

<i>WrongDimension</i>	Number of rows in the list bars_x needs to be the same of the length of ranges_y - 1.
<i>InconsistentGeometry</i>	The sum of the distances for each row in bars_x should be equal to the section's width (tol = 5 mm).
<i>InconsistentGeometry</i>	The sum of the distances in ranges_y should be equal to the section's depth (tol = 5 mm).
<i>InconsistentGeometry</i>	e should be smaller than half the depth and the width of the section.
<i>WrongDimension</i>	discr_core has a length of 2.
<i>WrongDimension</i>	discr_cover_lateral has a length of 2.
<i>WrongDimension</i>	discr_cover_topbottom has a length of 2.
<i>NegativeValue</i>	GJ needs to be positive.

Reimplemented in [FibersRectRCRectShape](#).

Definition at line 35 of file [Fibers.py](#).

```

00036     bars_x: np.ndarray, ranges_y: np.ndarray, discr_core: list, discr_cover_lateral: list,
        discr_cover_topbottom: list, GJ = 0.0):
00037     """
00038     Constructor of the class.
00039
00040     @param ID (int): Unique fiber section ID.
00041     @param b (float): Width of the section.
00042     @param d (float): Depth of the section.
00043     @param Ay (float): Area of one vertical reinforcing bar.
00044     @param D_hoops (float): Diameter of the hoops.
00045     @param e (float): Concrete cover.
00046     @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
        fibers.
00047     @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00048     @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
        fibers.
00049     @param bars_x (np.ndarray): Array with a range of aligned vertical reinforcing bars for each
        row in x direction.
00050         Distances from border to bar centerline, bar to bar centerlines and finally bar centerline
        to border in the x direction (aligned).
00051         Starting from the left to right, from the top range to the bottom one.
00052         The number of bars for each range can vary; in this case, add this argument when defining
        the array " dtype = object"
00053     @param ranges_y (np.ndarray): Array of dimension 1 with the position or spacing in y of the
        ranges in bars_x.
00054         Distances from border to range centerlines, range to range centerlines and finally range
        centerline to border in the y direction.
00055         Starting from the top range to the bottom one.
00056     @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
        confined core.
00057     @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK
        (y) for the lateral unconfined cover.
00058     @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK
        (y) for the top and bottom unconfined cover.
00059     @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
        Defaults to 0.0, assume no torsional stiffness.
00060
00061     @exception NegativeValue: ID needs to be a positive integer.
00062     @exception NegativeValue: b needs to be positive.
00063     @exception NegativeValue: d needs to be positive.
00064     @exception NegativeValue: Ay needs to be positive.
00065     @exception NegativeValue: D_hoops needs to be positive.
00066     @exception NegativeValue: e needs to be positive.
00067     @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
00068     @exception NegativeValue: conf_mat_ID needs to be a positive integer.
00069     @exception NegativeValue: bars_mat_ID needs to be a positive integer.
00070     @exception WrongDimension: Number of rows in the list bars_x needs to be the same of the
        length of ranges_y - 1.
00071     @exception InconsistentGeometry: The sum of the distances for each row in bars_x should be
        equal to the section's width (tol = 5 mm).
00072     @exception InconsistentGeometry: The sum of the distances in ranges_y should be equal to the
        section's depth (tol = 5 mm).
00073     @exception InconsistentGeometry: e should be smaller than half the depth and the width of the
        section.
00074     @exception WrongDimension: discr_core has a length of 2.
00075     @exception WrongDimension: discr_cover_lateral has a length of 2.
00076     @exception WrongDimension: discr_cover_topbottom has a length of 2.
00077     @exception NegativeValue: GJ needs to be positive.
00078     """

```

```

00079         # Check
00080         if ID < 1: raise NegativeValue()
00081         if b < 0: raise NegativeValue()
00082         if d < 0: raise NegativeValue()
00083         if Ay < 0: raise NegativeValue()
00084         if D_hoops < 0: raise NegativeValue()
00085         if e < 0: raise NegativeValue()
00086         if unconf_mat_ID < 1: raise NegativeValue()
00087         if conf_mat_ID < 1: raise NegativeValue()
00088         if bars_mat_ID < 1: raise NegativeValue()
00089         if np.size(bars_x) != np.size(ranges_y)-1: raise WrongDimension()
00090         geometry_tol = 5*mm_unit
00091         for bars in bars_x:
00092             if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
00093             if abs(np.sum(ranges_y) - d) > geometry_tol: raise InconsistentGeometry()
00094             if e > b/2 or e > d/2: raise InconsistentGeometry()
00095             if len(discr_core) != 2: raise WrongDimension()
00096             if len(discr_cover_lateral) != 2: raise WrongDimension()
00097             if len(discr_cover_topbottom) != 2: raise WrongDimension()
00098             if GJ < 0: raise NegativeValue()
00099
00100         # Arguments
00101         self.ID = ID
00102         self.b = b
00103         self.d = d
00104         self.Ay = Ay
00105         self.D_hoops = D_hoops
00106         self.e = e
00107         self.unconf_mat_ID = unconf_mat_ID
00108         self.conf_mat_ID = conf_mat_ID
00109         self.bars_mat_ID = bars_mat_ID
00110         self.bars_x = deepcopy(bars_x)
00111         self.ranges_y = copy(ranges_y)
00112         self.discr_core = copy(discr_core)
00113         self.discr_cover_lateral = copy(discr_cover_lateral)
00114         self.discr_cover_topbottom = copy(discr_cover_topbottom)
00115         self.GJ = GJ
00116
00117         # Initialized the parameters that are dependent from others
00118         self.section_name_tag = "None"
00119         self.Initialized = False
00120         self.ReInit()
00121

```

## 7.14.3 Member Function Documentation

### 7.14.3.1 CreateFibers()

```

def CreateFibers (
    self )

```

Method that initialises the fiber by calling the OpenSeesPy commands.

Definition at line 226 of file [Fibers.py](#).

```

00226     def CreateFibers(self):
00227         """
00228         Method that initialises the fiber by calling the OpenSeesPy commands.
00229         """
00230         create_fiber_section(self.fib_sec)
00231         self.Initialized = True
00232         self.UpdateStoredData()
00233
00234

```



### 7.14.3.2 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 122 of file `Fibers.py`.

```
00122     def ReInit(self):
00123         """
00124         Implementation of the homonym abstract method.
00125         See parent class DataManagement for detailed information.
00126         """
00127         # Memebers
00128         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"

00129
00130         # Parameters
00131         z1 = self.b/2
00132         y1 = self.d/2
00133         zc = z1-self.e-self.D_hoops/2
00134         yc = y1-self.e-self.D_hoops/2
00135
00136         # Create the concrete core fibers
00137         core = [-yc, -zc, yc, zc]
00138         core_cmd = ['patch', 'rect', self.conf_mat_ID, *self.dscr_core, *core]
00139
00140         # Create the concrete cover fibers (bottom left, top right)
00141         cover_up = [yc, -z1, y1, z1]
00142         cover_down = [-y1, -z1, -yc, z1]
00143         cover_left = [-yc, zc, yc, z1]
00144         cover_right = [-yc, -z1, yc, -zc]
00145         cover_up_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.dscr_cover_topbottom, *cover_up]
00146         cover_down_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.dscr_cover_topbottom,
*cover_down]
00147         cover_left_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.dscr_cover_lateral, *cover_left]
00148         cover_right_cmd = ['patch', 'rect', self.unconf_mat_ID, *self.dscr_cover_lateral,
*cover_right]
00149         self.fib_sec = [['section', 'Fiber', self.ID, '-GJ', self.GJ],
core_cmd, cover_up_cmd, cover_down_cmd, cover_left_cmd, cover_right_cmd]
00150
00151
00152         # Create the reinforcing fibers (top, middle, bottom)
00153         # NB: note the order of definition of bars_x and ranges_y
00154         nr_bars = 0
00155         for range in self.bars_x:
00156             nr_bars += np.size(range)-1
00157         rebarY = -np.cumsum(self.ranges_y[0:-1]) + y1
00158         self.rebarYZ = np.zeros((nr_bars, 2))
00159
00160         iter = 0
00161         for ii, Y in enumerate(rebarY):
00162             rebarZ = -np.cumsum(self.bars_x[ii][0:-1]) + z1
00163             for Z in rebarZ:
00164                 self.rebarYZ[iter, :] = [Y, Z]
00165                 iter = iter + 1
00166
00167         for YZ in self.rebarYZ:
00168             self.fib_sec.append(['layer', 'bar', self.bars_mat_ID, self.Ay, *YZ])
00169
00170         # Data storage for loading/saving
00171         self.UpdateStoredData()
00172
00173
```

### 7.14.3.3 ShowInfo()

```
def ShowInfo (
    self,
    plot = False,
    block = False )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

## Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the fiber. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 199 of file [Fibers.py](#).

```

00199     def ShowInfo(self, plot = False, block = False):
00200         """
00201         Implementation of the homonym abstract method.
00202         See parent class DataManagement for detailed information.
00203
00204         @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
00205         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00206         """
00207         print("")
00208         print("Requested info for FibersRect, ID = {}".format(self.ID))
00209         print("Section associated: {}".format(self.section_name_tag))
00210         print("Base b = {} mm and depth d = {} mm".format(self.b/mm_unit, self.d/mm_unit))
00211         print("Confined material model ID = {}".format(self.conf_mat_ID))
00212         print("Unconfined material model ID = {}".format(self.unconf_mat_ID))
00213         print("Bars material model ID = {}".format(self.bars_mat_ID))
00214         print("Discretisation in the core [IJ or x/z dir, JK or y dir] = {}".format(self.discr_core))
00215         print("Discretisation in the lateral covers [IJ or x/z dir, JK or y dir] =
{}".format(self.discr_cover_lateral))
00216         print("Discretisation in the top and bottom covers [IJ or x/z dir, JK or y dir] =
{}".format(self.discr_cover_topbottom))
00217         print("")
00218
00219         if plot:
00220             plot_fiber_section(self.fib_sec, matcolor=['#808080', '#D3D3D3', 'k'])
00221
00222         if block:
00223             plt.show()
00224
00225

```

#### 7.14.3.4 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 175 of file [Fibers.py](#).

```

00175     def UpdateStoredData(self):
00176         """
00177         Implementation of the homonym abstract method.
00178         See parent class DataManagement for detailed information.
00179         """
00180         self.data = [{"INFO_TYPE", "FibersRect"}, # Tag for differentiating different data
00181                     ["ID", self.ID],
00182                     ["section_name_tag", self.section_name_tag],
00183                     ["b", self.b],
00184                     ["d", self.d],
00185                     ["Ay", self.Ay],
00186                     ["D_hoops", self.D_hoops],
00187                     ["e", self.e],
00188                     ["GJ", self.GJ],
00189                     ["conf_mat_ID", self.conf_mat_ID],
00190                     ["discr_core", self.discr_core],
00191                     ["unconf_mat_ID", self.unconf_mat_ID],
00192                     ["discr_cover_topbottom", self.discr_cover_topbottom],
00193                     ["discr_cover_lateral", self.discr_cover_lateral],
00194                     ["bars_mat_ID", self.bars_mat_ID],
00195                     ["bars_x", self.bars_x],
00196                     ["ranges_y", self.ranges_y],
00197                     ["Initialized", self.Initialized]]
00198

```

## 7.14.4 Member Data Documentation

### 7.14.4.1 Ay

Ay

Definition at line 104 of file [Fibers.py](#).

### 7.14.4.2 b

b

Definition at line 102 of file [Fibers.py](#).

### 7.14.4.3 bars\_mat\_ID

bars\_mat\_ID

Definition at line 109 of file [Fibers.py](#).

### 7.14.4.4 bars\_x

bars\_x

Definition at line 110 of file [Fibers.py](#).

### 7.14.4.5 conf\_mat\_ID

conf\_mat\_ID

Definition at line 108 of file [Fibers.py](#).

#### 7.14.4.6 **d**

`d`

Definition at line 103 of file [Fibers.py](#).

#### 7.14.4.7 **D\_hoops**

`D_hoops`

Definition at line 105 of file [Fibers.py](#).

#### 7.14.4.8 **data**

`data`

Definition at line 180 of file [Fibers.py](#).

#### 7.14.4.9 **discr\_core**

`discr_core`

Definition at line 112 of file [Fibers.py](#).

#### 7.14.4.10 **discr\_cover\_lateral**

`discr_cover_lateral`

Definition at line 113 of file [Fibers.py](#).

#### 7.14.4.11 **discr\_cover\_topbottom**

`discr_cover_topbottom`

Definition at line 114 of file [Fibers.py](#).

#### 7.14.4.12 e

e

Definition at line 106 of file [Fibers.py](#).

#### 7.14.4.13 fib\_sec

fib\_sec

Definition at line 149 of file [Fibers.py](#).

#### 7.14.4.14 GJ

GJ

Definition at line 115 of file [Fibers.py](#).

#### 7.14.4.15 ID

ID

Definition at line 101 of file [Fibers.py](#).

#### 7.14.4.16 Initialized

Initialized

Definition at line 119 of file [Fibers.py](#).

#### 7.14.4.17 ranges\_y

ranges\_y

Definition at line 111 of file [Fibers.py](#).

#### 7.14.4.18 rebarYZ

rebarYZ

Definition at line 158 of file [Fibers.py](#).

#### 7.14.4.19 section\_name\_tag

section\_name\_tag

Definition at line 118 of file [Fibers.py](#).

#### 7.14.4.20 unconf\_mat\_ID

unconf\_mat\_ID

Definition at line 107 of file [Fibers.py](#).

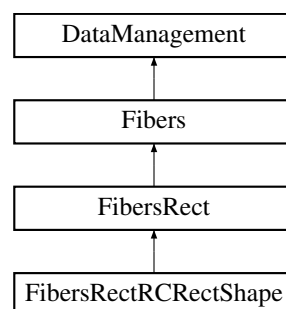
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py](#)

## 7.15 FibersRectRCRectShape Class Reference

Class that is the children of [FibersRect](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.

Inheritance diagram for [FibersRectRCRectShape](#):



### Public Member Functions

- `def __init__(self, int ID, RCRectShape section, int unconf_mat_ID, int conf_mat_ID, int bars_mat_ID, list discr_core, list discr_cover_lateral, list discr_cover_topbottom, GJ=0)`

*Constructor of the class.*

## Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.15.1 Detailed Description

Class that is the children of [FibersRect](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.

#### Parameters

<a href="#">FibersRect</a>	Parent class.
----------------------------	---------------

Definition at line 235 of file [Fibers.py](#).

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RCRectShape section,
    int unconf_mat_ID,
    int conf_mat_ID,
    int bars_mat_ID,
    list discr_core,
    list discr_cover_lateral,
    list discr_cover_topbottom,
    GJ = 0 )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique fiber section ID.
<i>section</i>	(RCRectShape): RCRectShape section object.
<i>unconf_mat_ID</i>	(int): ID of material model that will be assigned to the unconfined fibers.
<i>conf_mat_ID</i>	(int): ID of material model that will be assigned to the confined fibers.
<i>bars_mat_ID</i>	(int): ID of material model that will be assigned to the reinforcing bars fibers.
<i>discr_core</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the confined core.
<i>discr_cover_lateral</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the lateral unconfined core.
<i>discr_cover_topbottom</i>	(list): List with two entries: discretisation in IJ (x/z) and JK (y) for the top and bottom unconfined core.
<i>GJ</i>	(float, optional): Linear-elastic torsional stiffness assigned to the section. Defaults to 0.0, assume no torsional stiffness.

Reimplemented from [FibersRect](#).

Definition at line 241 of file [Fibers.py](#).

```

00242         discr_core: list, discr_cover_lateral: list, discr_cover_topbottom: list, GJ=0):
00243         """
00244         Constructor of the class.
00245
00246         @param ID (int): Unique fiber section ID.
00247         @param section (RCRectShape): RCRectShape section object.
00248         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
00249         fibers.
00249         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00250         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
00251         fibers.
00251         @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
00252         confined core.
00252         @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK
00253         (y) for the lateral unconfined core.
00253         @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK
00254         (y) for the top and bottom unconfined core.
00254         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
00255         Defaults to 0.0, assume no torsional stiffness.
00255         """
00256         self.section = deepcopy(section)
00257         super().__init__(ID, section.b, section.d, section.Ay, section.D_hoops, section.e,
00258         unconf_mat_ID, conf_mat_ID, bars_mat_ID,
00258         section.bars_position_x, section.bars_ranges_position_y, discr_core, discr_cover_lateral,
00259         discr_cover_topbottom, GJ=GJ)
00259         self.section_name_tag = section.name_tag
00260         self.UpdateStoredData()
00261
00262

```

## 7.15.3 Member Data Documentation

### 7.15.3.1 section

section

Definition at line 256 of file [Fibers.py](#).

### 7.15.3.2 section\_name\_tag

section\_name\_tag

Definition at line 259 of file [Fibers.py](#).

The documentation for this class was generated from the following file:

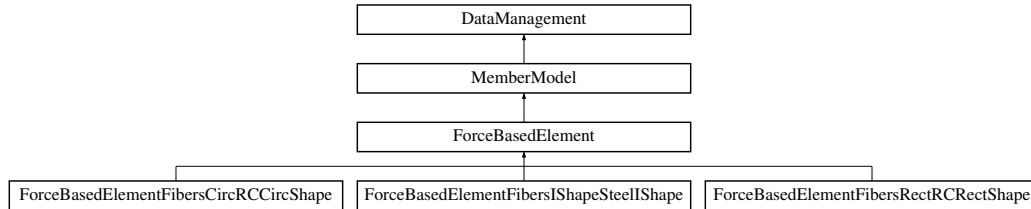
- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[Fibers.py](#)



## 7.16 ForceBasedElement Class Reference

Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for ForceBasedElement:



### Public Member Functions

- `def __init__ (self, int iNode_ID, int jNode_ID, int fiber_ID, int geo_transf_ID, new_integration_ID=-1, lp=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID=-1)`  
*Constructor of the class.*
- `def CreateMember (self)`  
*Method that initialises the member by calling the OpenSeesPy commands through various functions.*
- `def Record (self, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)`  
*Implementation of the homonym abstract method.*
- `def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)`  
*Implementation of the homonym abstract method.*
- `def Relnit (self, new_integration_ID, ele_ID=-1)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

### Public Attributes

- `data`
- `element_array`
- `element_ID`
- `fiber_ID`
- `geo_transf_ID`
- `Initialized`
- `iNode_ID`
- `integration_type`
- `lp`
- `jNode_ID`
- `max_iter`
- `new_integration_ID`
- `section_name_tag`
- `tol`

#### 7.16.1 Detailed Description

Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

## Parameters

<a href="#">MemberModel</a>	Parent abstract class.
-----------------------------	------------------------

Definition at line 964 of file [MemberModel.py](#).

## 7.16.2 Constructor & Destructor Documentation

### 7.16.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    int fiber_ID,
    int geo_transf_ID,
    new_integration_ID = -1,
    Ip = 5,
    integration_type = "Lobatto",
    max_iter = MAX_ITER_INTEGRATION,
    tol = TOL_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber_ID</i>	(int): ID of the fiber section.
<i>geo_transf_ID</i>	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">Relnit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>integration_type</i>	(str, optional): Integration type. FOR more information, see OpenSeesPy documentation. Defaults to "Lobatto".
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>tol</i>	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer, if different from -1.
<i>NegativeValue</i>	Ip needs to be a positive integer bigger than 3, if different from -1.
<i>NegativeValue</i>	max_iter needs to be a positive integer.
<i>NegativeValue</i>	tol needs to be positive.
<i>NegativeValue</i>	ID needs to be a positive integer, if different from -1.

Reimplemented in [ForceBasedElementFibersCircRCCircShape](#), [ForceBasedElementFibersIShapeSteelShape](#), and [ForceBasedElementFibersRectRCRectShape](#).

Definition at line 970 of file [MemberModel.py](#).

```

00971         new_integration_ID = -1, Ip = 5, integration_type = "Lobatto", max_iter =
MAX_ITER_INTEGRATION, tol = TOL_INTEGRATION, ele_ID = -1):
00972         """
00973         Constructor of the class.
00974
00975         @param iNode_ID (int): ID of the first end node.
00976         @param jNode_ID (int): ID of the second end node.
00977         @param fiber_ID (int): ID of the fiber section.
00978         @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
00979         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
00980         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
00981         @param integration_type (str, optional): Integration type. For more information, see
OpenSeesPy documentation.
00982         Defaults to "Lobatto".
00983         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
00984         @param tol (float, optional): Tolerance for the integration convergence. Defaults to
TOL_INTEGRATION (Units).
00985         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00986
00987         @exception NegativeValue: ID needs to be a positive integer.
00988         @exception NegativeValue: ID needs to be a positive integer.
00989         @exception NegativeValue: ID needs to be a positive integer.
00990         @exception NegativeValue: ID needs to be a positive integer.
00991         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00992         @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from
-1.
00993
00994         @exception NegativeValue: max_iter needs to be a positive integer.
00995         @exception NegativeValue: tol needs to be positive.
00996         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00997         """
00998         # Check
00999         if iNode_ID < 1: raise NegativeValue()
01000         if jNode_ID < 1: raise NegativeValue()
01001         if fiber_ID < 1: raise NegativeValue()
01002         if geo_transf_ID < 1: raise NegativeValue()
01003         if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
01004         if Ip != -1 and Ip < 3: raise NegativeValue()
01005         if max_iter < 0: raise NegativeValue()
01006         if tol < 0: raise NegativeValue()
01007         if ele_ID != -1 and ele_ID < 1: raise NegativeValue()
01008
01009         # Arguments
01010         self.iNode_ID = iNode_ID
01011         self.jNode_ID = jNode_ID
01012         self.fiber_ID = fiber_ID
01013         self.geo_transf_ID = geo_transf_ID
01014         self.Ip = Ip
01015         self.integration_type = integration_type
01016         self.max_iter = max_iter
01017         self.tol = tol
01018
01019         # Initialized the parameters that are dependent from others
01020         self.section_name_tag = "None"
01021         self.Initialized = False
01022         self.ReInit(new_integration_ID, ele_ID)
01023

```

## 7.16.3 Member Function Documentation

### 7.16.3.1 CreateMember()

```
def CreateMember (
    self )
```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 1092 of file [MemberModel.py](#).

```
01092     def CreateMember(self):
01093         """
01094         Method that initialises the member by calling the OpenSeesPy commands through various
01095         functions.
01096         """
01097         self.element_array = [[self.element_ID, self.iNode_ID, self.jNode_ID]]
01098         # Define integration type
01099         beamIntegration(self.integration_type, self.new_integration_ID, self.fiber_ID, self.Ip)
01100
01101         # Define element
01102         element('forceBeamColumn', self.element_ID, self.iNode_ID, self.jNode_ID, self.geo_transf_ID,
01103               self.new_integration_ID, '-iter', self.max_iter, self.tol)
01104
01105         # Update class
01106         self.Initialized = True
01107         self.UpdateStoredData()
01108
```

### 7.16.3.2 Record()

```
def Record (
    self,
    str name_txt,
    str data_dir,
    force_rec = True,
    def_rec = True,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 1109 of file [MemberModel.py](#).

```
01109     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
01110         """
01111         Implementation of the homonym abstract method.
01112         See parent class MemberModel for detailed information.
01113         """
01114         super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
01115               time_rec=time_rec)
01116
```

### 7.16.3.3 RecordNodeDef()

```
def RecordNodeDef (
    self,
    str name_txt,
    str data_dir,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 1117 of file [MemberModel.py](#).

```
01117     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
01118         """
01119         Implementation of the homonym abstract method.
01120         See parent class MemberModel for detailed information.
01121         """
01122         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
01123
01124
```

### 7.16.3.4 ReInit()

```
def ReInit (
    self,
    new_integration_ID,
    ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

#### Parameters

<i>new_integration_ID</i>	(int): ID of the integration technique.
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use <a href="#">IDConvention</a> to define it.

Definition at line 1024 of file [MemberModel.py](#).

```
01024     def ReInit(self, new_integration_ID, ele_ID = -1):
01025         """
01026         Implementation of the homonym abstract method.
01027         See parent class DataManagement for detailed information.
01028
01029         @param new_integration_ID (int): ID of the integration technique.
01030         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01031         IDConvention to define it.
01032         """
01033         # Precompute some members
01034         self.element_ID = IDConvention(self.iNode_ID, self.jNode_ID) if ele_ID == -1 else ele_ID
01035
01036         # Arguments
01037         self.new_integration_ID = self.element_ID if new_integration_ID == -1 else new_integration_ID
01038
01039         # Members
01040         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
01041
01042         # Data storage for loading/saving
```

```

01042         self.UpdateStoredData()
01043
01044

```

### 7.16.3.5 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 1066 of file [MemberModel.py](#).

```

01066     def ShowInfo(self, plot = False, block = False):
01067         """
01068         Implementation of the homonym abstract method.
01069         See parent class DataManagement for detailed information.
01070
01071         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
01072         False.
01073         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
01074         of the program everytime that a plot should pop up). Defaults to False.
01075         """
01076         print("")
01077         print("Requested info for ForceBasedElement member model, ID = {}".format(self.element_ID))
01078         print("Fiber associated, ID = {}".format(self.fiber_ID))
01079         print("Integration type '{}', ID = {}".format(self.integration_type, self.new_integration_ID))
01080         print("Section associated {}".format(self.section_name_tag))
01081         print("Number of integration points along the element Ip = {}, max iter = {}, tol =
01082         {}".format(self.Ip, self.max_iter, self.tol))
01083         print("Geometric transformation = {}".format(self.geo_transf_ID))
01084         print("")
01085         if plot:
01086             if self.Initialized:
01087                 plot\_member(self.element_array, "ForceBased Element, ID = {}".format(self.element_ID))
01088                 if block:
01089                     plt.show()
01090             else:
01091                 print("The ForceBasedElement is not initialized (element not created), ID =
01092                 {}".format(self.element_ID))
01093
01094
01095

```

### 7.16.3.6 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 1046 of file `MemberModel.py`.

```
01046     def UpdateStoredData(self):
01047         """
01048         Implementation of the homonym abstract method.
01049         See parent class DataManagement for detailed information.
01050         """
01051         self.data = [{"INFO_TYPE", "ForceBasedElement"}, # Tag for differentiating different data
01052                     ["element_ID", self.element_ID],
01053                     ["section_name_tag", self.section_name_tag],
01054                     ["Ip", self.Ip],
01055                     ["iNode_ID", self.iNode_ID],
01056                     ["jNode_ID", self.jNode_ID],
01057                     ["fiber_ID", self.fiber_ID],
01058                     ["new_integration_ID", self.new_integration_ID],
01059                     ["integration_type", self.integration_type],
01060                     ["tol", self.tol],
01061                     ["max_iter", self.max_iter],
01062                     ["transf_ID", self.geo_transf_ID],
01063                     ["Initialized", self.Initialized]]
01064
01065
```

## 7.16.4 Member Data Documentation

### 7.16.4.1 data

data

Definition at line 1051 of file `MemberModel.py`.

### 7.16.4.2 element\_array

element\_array

Definition at line 1096 of file `MemberModel.py`.

### 7.16.4.3 element\_ID

element\_ID

Definition at line 1033 of file `MemberModel.py`.

### 7.16.4.4 fiber\_ID

fiber\_ID

Definition at line 1011 of file `MemberModel.py`.

#### 7.16.4.5 geo\_transf\_ID

geo\_transf\_ID

Definition at line 1012 of file [MemberModel.py](#).

#### 7.16.4.6 Initialized

Initialized

Definition at line 1020 of file [MemberModel.py](#).

#### 7.16.4.7 iNode\_ID

iNode\_ID

Definition at line 1009 of file [MemberModel.py](#).

#### 7.16.4.8 integration\_type

integration\_type

Definition at line 1014 of file [MemberModel.py](#).

#### 7.16.4.9 Ip

Ip

Definition at line 1013 of file [MemberModel.py](#).

#### 7.16.4.10 jNode\_ID

jNode\_ID

Definition at line 1010 of file [MemberModel.py](#).



#### 7.16.4.11 max\_iter

`max_iter`

Definition at line 1015 of file [MemberModel.py](#).

#### 7.16.4.12 new\_integration\_ID

`new_integration_ID`

Definition at line 1036 of file [MemberModel.py](#).

#### 7.16.4.13 section\_name\_tag

`section_name_tag`

Definition at line 1019 of file [MemberModel.py](#).

#### 7.16.4.14 tol

`tol`

Definition at line 1016 of file [MemberModel.py](#).

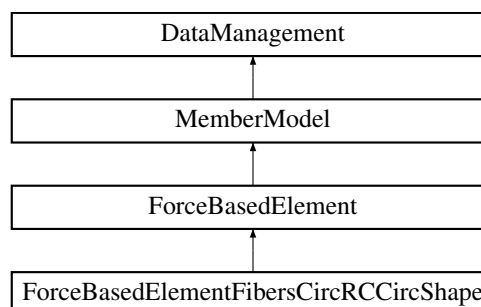
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.17 ForceBasedElementFibersCircRCCircShape Class Reference

Class that is the children of [ForceBasedElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.

Inheritance diagram for [ForceBasedElementFibersCircRCCircShape](#):



## Public Member Functions

- `def __init__ (self, int iNode_ID, int jNode_ID, FibersCircRCCircShape fiber, int geo_transf_ID, new_integration_ID=-1, lp=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID=-1)`

*Constructor of the class.*

## Public Attributes

- `section`
- `section_name_tag`

### 7.17.1 Detailed Description

Class that is the children of [ForceBasedElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.

#### Parameters

<a href="#">ForceBasedElement</a>	Parent class.
-----------------------------------	---------------

Definition at line 1157 of file [MemberModel.py](#).

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    FibersCircRCCircShape fiber,
    int geo_transf_ID,
    new_integration_ID = -1,
    lp = 5,
    integration_type = "Lobatto",
    max_iter = MAX_ITER_INTEGRATION,
    tol = TOL_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

#### Parameters

<code>iNode_ID</code>	(int): ID of the first end node.
-----------------------	----------------------------------

## Parameters

<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber</i>	(FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>integration_type</i>	(str, optional): Integration type. For more information, see OpenSeesPy documentation. Defaults to "Lobatto".
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integration convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>tol</i>	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [ForceBasedElement](#).

Definition at line 1163 of file [MemberModel.py](#).

```

01164         new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
        tol=TOL_INTEGRATION, ele_ID = -1):
01165     """
01166     Constructor of the class.
01167
01168     @param iNode_ID (int): ID of the first end node.
01169     @param jNode_ID (int): ID of the second end node.
01170     @param fiber (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
01171     @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
01172     @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01173     @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01174     @param integration_type (str, optional): Integration type. For more information, see
OpenSeesPy documentation.
        Defaults to "Lobatto".
01175     @param max_iter (int, optional): Maximal number of iteration to reach the integration
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01176     @param tol (float, optional): Tolerance for the integration convergence. Defaults to
TOL_INTEGRATION (Units).
01177     @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01178     """
01179     self.section = deepcopy(fiber.section)
01180     super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01181                     new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
max_iter=max_iter, tol=tol, ele_ID=ele_ID)
01182     self.section_name_tag = self.section.name_tag
01183     self.UpdateStoredData()
01184     # Check length
01185     self._CheckL()
01186
01187
01188

```

## 7.17.3 Member Data Documentation

### 7.17.3.1 section

section

Definition at line 1180 of file [MemberModel.py](#).

### 7.17.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1183 of file [MemberModel.py](#).

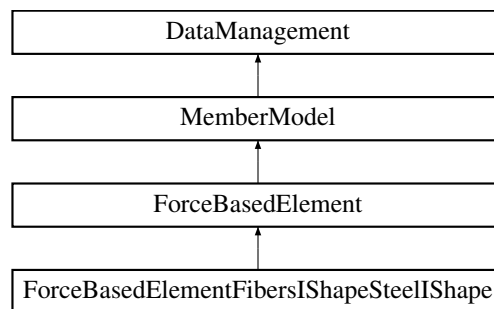
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.18 ForceBasedElementFibersIShapeSteelIShape Class Reference

Class that is the children of [ForceBasedElement](#) and combine the class [FibersIShapeSteelIShape](#) (fiber section) to retrieve the information needed.

Inheritance diagram for [ForceBasedElementFibersIShapeSteelIShape](#):



### Public Member Functions

- `def __init__ (self, int iNode_ID, int jNode_ID, FibersIShapeSteelIShape fiber, int geo\_transf\_ID, new\_integration\_ID=-1, lp=5, integration\_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID=-1)`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.18.1 Detailed Description

Class that is the children of [ForceBasedElement](#) and combine the class [FibersIShapeSteelIShape](#) (fiber section) to retrieve the information needed.

## Parameters

<a href="#">ForceBasedElement</a>	Parent class.
-----------------------------------	---------------

Definition at line 1189 of file [MemberModel.py](#).

## 7.18.2 Constructor & Destructor Documentation

### 7.18.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    FibersIShapeSteelIShape fiber,
    int geo_transf_ID,
    new_integration_ID = -1,
    Ip = 5,
    integration_type = "Lobatto",
    max_iter = MAX_ITER_INTEGRATION,
    tol = TOL_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber</i>	(FibersIShapeSteelIShape): FibersIShapeSteelIShape fiber section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">Relnit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>integration_type</i>	(str, optional): Integration type. For more information, see OpenSeesPy documentation. Defaults to "Lobatto".
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integration convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>tol</i>	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [ForceBasedElement](#).

Definition at line 1195 of file [MemberModel.py](#).

```
01196     new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
        tol=TOL_INTEGRATION, ele_ID = -1):
01197     """
```

```

01198         Constructor of the class.
01199
01200         @param iNode_ID (int): ID of the first end node.
01201         @param jNode_ID (int): ID of the second end node.
01202         @param fiber (FibersIShapeSteelIShape): FibersIShapeSteelIShape fiber section object.
01203         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
01204         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01205         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01206         @param integration_type (str, optional): Integration type. For more information, see
OpenSeesPy documentation.
01207         Defaults to "Lobatto".
01208         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01209         @param tol (float, optional): Tolerance for the integration convergence. Defaults to
TOL_INTEGRATION (Units).
01210         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01211         """
01212         self.section = deepcopy(fiber.section)
01213         super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01214             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
max_iter=max_iter, tol=tol, ele_ID=ele_ID)
01215         self.section_name_tag = self.section.name_tag
01216         self.UpdateStoredData()
01217         # Check length
01218         self._CheckL()
01219
01220

```

## 7.18.3 Member Data Documentation

### 7.18.3.1 section

section

Definition at line 1212 of file [MemberModel.py](#).

### 7.18.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1215 of file [MemberModel.py](#).

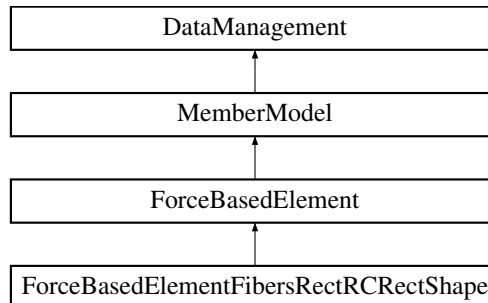
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.19 ForceBasedElementFibersRectRCRectShape Class Reference

Class that is the children of [ForceBasedElement](#) and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

Inheritance diagram for ForceBasedElementFibersRectRCRectShape:



### Public Member Functions

- `def __init__ (self, int iNode_ID, int jNode_ID, FibersRectRCRectShape fiber, int geo_transf_ID, new_integration_ID=-1, lp=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION, tol=TOL_INTEGRATION, ele_ID=-1)`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.19.1 Detailed Description

Class that is the children of [ForceBasedElement](#) and combine the class FibersRectRCRectShape (fiber section) to retrieve the information needed.

#### Parameters

<a href="#">ForceBasedElement</a>	Parent class.
-----------------------------------	---------------

Definition at line 1125 of file [MemberModel.py](#).

#### 7.19.2 Constructor & Destructor Documentation

### 7.19.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    FibersRectRCRectShape fiber,
    int geo_transf_ID,
    new_integration_ID = -1,
    Ip = 5,
    integration_type = "Lobatto",
    max_iter = MAX_ITER_INTEGRATION,
    tol = TOL_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

#### Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber</i>	(FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>integration_type</i>	(str, optional): Integration type. For more information, see OpenSeesPy documentation. Defaults to "Lobatto".
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integration convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>tol</i>	(float, optional): Tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [ForceBasedElement](#).

Definition at line 1131 of file [MemberModel.py](#).

```
01132     new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
    tol=TOL_INTEGRATION, ele_ID = -1):
01133     """
01134     Constructor of the class.
01135
01136     @param iNode_ID (int): ID of the first end node.
01137     @param jNode_ID (int): ID of the second end node.
01138     @param fiber (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
01139     @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
    documentation).
01140     @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
    e.g. computed in ReInit().
01141     @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01142     @param integration_type (str, optional): Integration type. For more information, see
    OpenSeesPy documentation.
    Defaults to "Lobatto".
01143     @param max_iter (int, optional): Maximal number of iteration to reach the integration
    convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01144     @param tol (float, optional): Tolerance for the integration convergence. Defaults to
    TOL_INTEGRATION (Units).
01145     @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
    IDConvention to define it.
01146     """
01147     self.section = deepcopy(fiber.section)
01148     super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
```



```

01150         new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
max_iter=max_iter, tol=tol, ele_ID= ele_ID)
01151         self.section_name_tag = self.section.name_tag
01152         self.UpdateStoredData()
01153         # Check length
01154         self._CheckL()
01155
01156

```

### 7.19.3 Member Data Documentation

#### 7.19.3.1 section

section

Definition at line 1148 of file [MemberModel.py](#).

#### 7.19.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1151 of file [MemberModel.py](#).

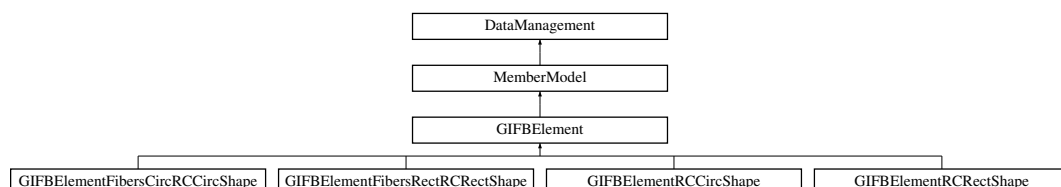
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.20 GIFBElement Class Reference

Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for GIFBElement:



## Public Member Functions

- def `__init__` (self, int `iNode_ID`, int `jNode_ID`, int `fiber_ID`, `D_bars`, `fy`, int `geo_transf_ID`, `lambda_i`=1, `lambda_j`=-1, `Lp`=-1, `lp`=-1, `new_integration_ID`=-1, `min_tol`=TOL\_INTEGRATION, `max_tol`=TOL\_INTEGRATION \* 1e4, `max_iter`=MAX\_ITER\_INTEGRATION, `ele_ID`=-1)  
*Constructor of the class.*
- def `Computelp` (self)  
*Compute the number of integration points with equal distance along the element.*
- def `ComputeLp` (self)  
*Method that computes the plastic length using Paulay 1992.*
- def `CreateMember` (self)  
*Method that initialises the member by calling the OpenSeesPy commands through various functions.*
- def `Record` (self, str `name_txt`, str `data_dir`, `force_rec`=True, `def_rec`=True, `time_rec`=True)  
*Implementation of the homonym abstract method.*
- def `RecordNodeDef` (self, str `name_txt`, str `data_dir`, `time_rec`=True)  
*Implementation of the homonym abstract method.*
- def `Relnit` (self, `lambda_i`=-1, `lambda_j`=-1, `Lp`=-1, `lp`=5, `new_integration_ID`=-1, `ele_ID`=-1)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, `plot`=False, `block`=False)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- `D_bars`
- `data`
- `element_array`
- `element_ID`
- `fiber_ID`
- `fy`
- `geo_transf_ID`
- `Initialized`
- `iNode_ID`
- `lp`
- `jNode_ID`
- `L`
- `lambda_i`
- `lambda_j`
- `Lp`
- `max_iter`
- `max_tol`
- `min_tol`
- `new_integration_ID`
- `section_name_tag`

### 7.20.1 Detailed Description

Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

The integration technique is Simpson. For more information, see Sideris and Salehi 2016, 2017 and 2020.

## Parameters

<a href="#">MemberModel</a>	Parent abstract class.
-----------------------------	------------------------

Definition at line 1221 of file [MemberModel.py](#).

## 7.20.2 Constructor & Destructor Documentation

### 7.20.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    int fiber_ID,
    D_bars,
    fy,
    int geo_transf_ID,
    lambda_i = -1,
    lambda_j = -1,
    Lp = -1,
    Ip = -1,
    new_integration_ID = -1,
    min_tol = TOL_INTEGRATION,
    max_tol = TOL_INTEGRATION*1e4,
    max_iter = MAX_ITER_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber_ID</i>	(int): ID of the fiber section.
<i>D_bars</i>	(float): Diameter of the vertical reinforcing bars.
<i>fy</i>	(float): Yield stress of the reinforcing bars.
<i>geo_transf_ID</i>	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
<i>lambda_i</i>	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
<i>lambda_j</i>	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
<i>Lp</i>	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>min_tol</i>	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).

## Parameters

<i>max_tol</i>	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	D_bars needs to be positive.
<i>NegativeValue</i>	fy needs to be positive.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	lambda_i needs to be positive.
<i>NegativeValue</i>	lambda_j needs to be positive.
<i>NegativeValue</i>	No plastic length defined.
<i>NegativeValue</i>	Lp needs to be positive, if different from -1.
<i>NegativeValue</i>	lp needs to be a positive integer bigger than 3, if different from -1.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	min_tol needs to be positive.
<i>NegativeValue</i>	max_tol needs to be positive.
<i>NegativeValue</i>	max_iter needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer, if different from -1.

Reimplemented in [GIFBElementFibersCircRCCircShape](#), [GIFBElementFibersRectRCRectShape](#), [GIFBElementRCCircShape](#), and [GIFBElementRCRectShape](#).

Definition at line 1229 of file [MemberModel.py](#).

```

01231     min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01232     ele_ID = -1):
01233     """
01234     Constructor of the class.
01235     @param iNode_ID (int): ID of the first end node.
01236     @param jNode_ID (int): ID of the second end node.
01237     @param fiber_ID (int): ID of the fiber section.
01238     @param D_bars (float): Diameter of the vertical reinforcing bars.
01239     @param fy (float): Yield stress of the reinforcing bars.
01240     @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
01241     @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01242         Defaults to -1, e.g. plastic hinge in the end i.
01243     @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01244         Defaults to -1, e.g. plastic hinge in the end j.
01245     @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01246     @param lp (int, optional): Number of integration points (min. 3). Defaults to 5.
01247     @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01248     @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION (Units).
01249     @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION*1e4.
01250     @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01251     @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01252

```

```

01253         @exception NegativeValue: ID needs to be a positive integer.
01254         @exception NegativeValue: ID needs to be a positive integer.
01255         @exception NegativeValue: ID needs to be a positive integer.
01256         @exception NegativeValue: D_bars needs to be positive.
01257         @exception NegativeValue: fy needs to be positive.
01258         @exception NegativeValue: ID needs to be a positive integer.
01259         @exception NegativeValue: lambda_i needs to be positive.
01260         @exception NegativeValue: lambda_j needs to be positive.
01261         @exception NegativeValue: No plastic length defined.
01262         @exception NegativeValue: Lp needs to be positive, if different from -1.
01263         @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from
-1.

01264         @exception NegativeValue: ID needs to be a positive integer.
01265         @exception NegativeValue: min_tol needs to be positive.
01266         @exception NegativeValue: max_tol needs to be positive.
01267         @exception NegativeValue: max_iter needs to be a positive integer.
01268         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
01269         """
01270         # Check
01271         if iNode_ID < 1: raise NegativeValue()
01272         if jNode_ID < 1: raise NegativeValue()
01273         if fiber_ID < 1: raise NegativeValue()
01274         if D_bars < 0: raise NegativeValue()
01275         if fy < 0: raise NegativeValue()
01276         if geo_transf_ID < 1: raise NegativeValue()
01277         if lambda_i != -1 and lambda_i < 0: raise NegativeValue()
01278         if lambda_j != -1 and lambda_j < 0: raise NegativeValue()
01279         if lambda_i == 0 and lambda_j == 0: print("!!!!!! WARNING !!!!!!! No plastic length defined
for element ID = {}".format(IDConvention(iNode_ID, jNode_ID)))
01280         if Lp != -1 and Lp < 0: raise NegativeValue()
01281         if Ip != -1 and Ip < 3: raise NegativeValue()
01282         if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
01283         if min_tol < 0: raise NegativeValue()
01284         if max_tol < 0: raise NegativeValue()
01285         if max_iter < 0: raise NegativeValue()
01286         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
01287
01288         # Arguments
01289         self.iNode_ID = iNode_ID
01290         self.jNode_ID = jNode_ID
01291         self.D_bars = D_bars
01292         self.fy = fy
01293         self.geo_transf_ID = geo_transf_ID
01294         self.fiber_ID = fiber_ID
01295         self.min_tol = min_tol
01296         self.max_tol = max_tol
01297         self.max_iter = max_iter
01298
01299         # Initialized the parameters that are dependent from others
01300         self.section_name_tag = "None"
01301         self.Initialized = False
01302         self.ReInit(lambda_i, lambda_j, Lp, Ip, new_integration_ID, ele_ID)
01303

```

## 7.20.3 Member Function Documentation

### 7.20.3.1 Computelp()

```

def ComputeIp (
    self )

```

Compute the number of integration points with equal distance along the element.

For more information, see Salehi and Sideris 2020.

**Returns**

int: Number of integration points

Definition at line 1439 of file [MemberModel.py](#).

```
01439     def ComputeIp(self):
01440         """
01441         Compute the number of integration points with equal distance along the element. For more
information, see Salehi and Sideris 2020.
01442
01443         @returns int: Number of integration points
01444         """
01445         tmp = math.ceil(1.5*self.L/self.Lp + 1)
01446         if (tmp % 2) == 0:
01447             return tmp + 1
01448         else:
01449             return tmp
01450
01451
```

**7.20.3.2 ComputeLp()**

```
def ComputeLp (
    self )
```

Method that computes the plastic length using Paulay 1992.

**Returns**

double: Plastic length

Definition at line 1430 of file [MemberModel.py](#).

```
01430     def ComputeLp(self):
01431         """
01432         Method that computes the plastic length using Paulay 1992.
01433
01434         @returns double: Plastic length
01435         """
01436         return (0.08*self.L/m_unit + 0.022*self.D_bars/m_unit*self.fy/MPa_unit)*m_unit
01437
01438
```

**7.20.3.3 CreateMember()**

```
def CreateMember (
    self )
```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 1396 of file [MemberModel.py](#).

```
01396     def CreateMember(self):
01397         """
01398         Method that initialises the member by calling the OpenSeesPy commands through various
functions.
01399         """
01400         self.element_array = [[self.element_ID, self.iNode_ID, self.jNode_ID]]
01401
01402         # Define integration type
01403         beamIntegration('Simpson', self.new_integration_ID, self.fiber_ID, self.Ip)
01404
01405         # Define element TODO: Dr. Salehi: lambda useless
01406         element('gradientInelasticBeamColumn', self.element_ID, self.iNode_ID, self.jNode_ID,
self.geo_transf_ID,
01407             self.new_integration_ID, self.lambda_i, self.lambda_j, self.Lp, '-iter', self.max_iter,
self.min_tol, self.max_tol)
01408
01409         # Update class
01410         self.Initialized = True
01411         self.UpdateStoredData()
01412
01413
```

### 7.20.3.4 Record()

```
def Record (
    self,
    str name_txt,
    str data_dir,
    force_rec = True,
    def_rec = True,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 1414 of file [MemberModel.py](#).

```
01414     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
01415         """
01416         Implementation of the homonym abstract method.
01417         See parent class MemberModel for detailed information.
01418         """
01419         super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
01420                        time_rec=time_rec)
01420
01421
```

### 7.20.3.5 RecordNodeDef()

```
def RecordNodeDef (
    self,
    str name_txt,
    str data_dir,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 1422 of file [MemberModel.py](#).

```
01422     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
01423         """
01424         Implementation of the homonym abstract method.
01425         See parent class MemberModel for detailed information.
01426         """
01427         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
01428
01429
```

### 7.20.3.6 ReInit()

```
def ReInit (
    self,
    lambda_i = -1,
    lambda_j = -1,
    Ip = -1,
    Ip = 5,
    new_integration_ID = -1,
    ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

## Parameters

<i>lambda_i</i>	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
<i>lambda_j</i>	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
<i>Lp</i>	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed here.
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use <a href="#">IDConvention</a> to define it.

Definition at line 1304 of file [MemberModel.py](#).

```

01304     def ReInit(self, lambda_i = -1, lambda_j = -1, Lp = -1, Ip = 5, new_integration_ID = -1, ele_ID =
-1):
01305         """
01306         Implementation of the homonym abstract method.
01307         See parent class DataManagement for detailed information.
01308
01309         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01310             Defaults to -1, e.g. plastic hinge in the end i.
01311         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01312             Defaults to -1, e.g. plastic hinge in the end j.
01313         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed here.
01314         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01315         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit\(\).
01316         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01317         """
01318         # Precompute some members
01319         iNode = np.array(nodeCoord(self.iNode_ID))
01320         jNode = np.array(nodeCoord(self.jNode_ID))
01321         self.L = np.linalg.norm(iNode-jNode)
01322         self.element_ID = IDConvention(self.iNode_ID, self.jNode_ID) if ele_ID == -1 else ele_ID
01323
01324         # Arguments
01325         self.Lp = self.ComputeLp() if Lp == -1 else Lp
01326         self.Ip = self.ComputeIp() if Ip == -1 else Ip
01327         self.lambda_i = self.Lp/self.L if lambda_i == -1 else lambda_i
01328         self.lambda_j = self.Lp/self.L if lambda_j == -1 else lambda_j
01329         self.new_integration_ID = self.element_ID if new_integration_ID == -1 else new_integration_ID
01330
01331         # Members
01332         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
01333
01334         # Data storage for loading/saving
01335         self.UpdateStoredData()
01336
01337

```

### 7.20.3.7 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.



## Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 1365 of file [MemberModel.py](#).

```

01365     def ShowInfo(self, plot = False, block = False):
01366         """
01367         Implementation of the homonym abstract method.
01368         See parent class DataManagement for detailed information.
01369
01370         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
01371         False.
01372         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
01373         of the program everytime that a plot should pop up). Defaults to False.
01374         """
01375         print("")
01376         print("Requested info for GIFBElement member model, ID = {}".format(self.element_ID))
01377         print("Fiber associated, ID = {}".format(self.fiber_ID))
01378         print("Integration type 'Simpson', ID = {}".format(self.new_integration_ID))
01379         print("Section associated {}".format(self.section_name_tag))
01380         print("Length L = {} m".format(self.L/m_unit))
01381         print("Diameter of the reinforcing bars DBars = {} mm".format(self.DBars/mm2_unit))
01382         print("Reinforcing bar steel strength fy = {} MPa".format(self.fy/MPa_unit))
01383         print("Plastic length Lp = {} mm".format(self.Lp/mm_unit))
01384         print("Number of integration points along the element Ip = {}, max iter = {}, (min, max tol) =
01385         ({}, {})".format(self.Ip, self.max_iter, self.min_tol, self.max_tol))
01386         print("Lambda_i = {} and lambda_j = {}".format(self.lambda_i, self.lambda_j))
01387         print("Geometric transformation = {}".format(self.geo_transf_ID))
01388         print("")
01389         if plot:
01390             if self.Initialized:
01391                 plot_member(self.element_array, "GIFB Element, ID = {}".format(self.element_ID))
01392                 if block:
01393                     plt.show()
01394             else:
01395                 print("The GIFBElement is not initialized (element not created), ID =
01396                 {}".format(self.element_ID))
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000

```

### 7.20.3.8 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 1339 of file [MemberModel.py](#).

```

1339     def UpdateStoredData(self):
1340         """
1341         Implementation of the homonym abstract method.
1342         See parent class DataManagement for detailed information.
1343         """
1344         self.data = [{"INFO_TYPE": "GIFBElement", # Tag for differentiating different data
1345                       ["element_ID", self.element_ID],
1346                       ["section_name_tag", self.section_name_tag],
1347                       ["L", self.L],
1348                       ["DBars", self.DBars],
1349                       ["fy", self.fy],
1350                       ["Lp", self.Lp],
1351                       ["Ip", self.Ip],
1352                       ["iNode_ID", self.iNode_ID],
1353                       ["lambda_i", self.lambda_i],
1354                       ["jNode_ID", self.jNode_ID],
1355                       ["lambda_j", self.lambda_j],
1356                       ["fiber_ID", self.fiber_ID],

```

```
01357         ["new_integration_ID", self.new_integration_ID],
01358         ["min_tol", self.min_tol],
01359         ["max_tol", self.max_tol],
01360         ["max_iter", self.max_iter],
01361         ["tranf_ID", self.geo_transf_ID],
01362         ["Initialized", self.Initialized]]
01363
01364
```

## 7.20.4 Member Data Documentation

### 7.20.4.1 D\_bars

D\_bars

Definition at line 1291 of file [MemberModel.py](#).

### 7.20.4.2 data

data

Definition at line 1344 of file [MemberModel.py](#).

### 7.20.4.3 element\_array

element\_array

Definition at line 1400 of file [MemberModel.py](#).

### 7.20.4.4 element\_ID

element\_ID

Definition at line 1322 of file [MemberModel.py](#).

### 7.20.4.5 fiber\_ID

fiber\_ID

Definition at line 1294 of file [MemberModel.py](#).

#### 7.20.4.6 fy

fy

Definition at line 1292 of file [MemberModel.py](#).

#### 7.20.4.7 geo\_transf\_ID

geo\_transf\_ID

Definition at line 1293 of file [MemberModel.py](#).

#### 7.20.4.8 Initialized

Initialized

Definition at line 1301 of file [MemberModel.py](#).

#### 7.20.4.9 iNode\_ID

iNode\_ID

Definition at line 1289 of file [MemberModel.py](#).

#### 7.20.4.10 Ip

Ip

Definition at line 1326 of file [MemberModel.py](#).

#### 7.20.4.11 jNode\_ID

jNode\_ID

Definition at line 1290 of file [MemberModel.py](#).

**7.20.4.12 L**

L

Definition at line [1321](#) of file [MemberModel.py](#).

**7.20.4.13 lambda\_i**

lambda\_i

Definition at line [1327](#) of file [MemberModel.py](#).

**7.20.4.14 lambda\_j**

lambda\_j

Definition at line [1328](#) of file [MemberModel.py](#).

**7.20.4.15 Lp**

Lp

Definition at line [1325](#) of file [MemberModel.py](#).

**7.20.4.16 max\_iter**

max\_iter

Definition at line [1297](#) of file [MemberModel.py](#).

**7.20.4.17 max\_tol**

max\_tol

Definition at line [1296](#) of file [MemberModel.py](#).

**7.20.4.18 min\_tol**`min_tol`

Definition at line 1295 of file [MemberModel.py](#).

**7.20.4.19 new\_integration\_ID**`new_integration_ID`

Definition at line 1329 of file [MemberModel.py](#).

**7.20.4.20 section\_name\_tag**`section_name_tag`

Definition at line 1300 of file [MemberModel.py](#).

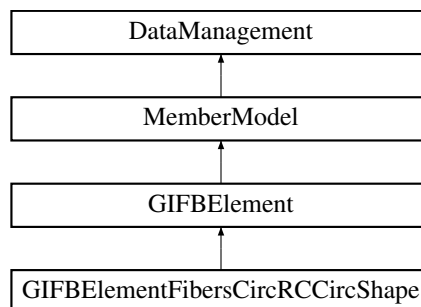
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

**7.21 GIFBElementFibersCircRCCircShape Class Reference**

Class that is the children of [GIFBElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.

Inheritance diagram for GIFBElementFibersCircRCCircShape:

**Public Member Functions**

- `def __init__(self, int iNode_ID, int jNode_ID, FibersCircRCCircShape fib, int geo_transf_ID, lambda_i=-1, lambda_j=-1, Lp=-1, lp=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL_INTEGRATION * 1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)`

*Constructor of the class.*

## Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.21.1 Detailed Description

Class that is the children of [GIFBElement](#) and combine the class `FibersCircRCCircShape` (fiber section) to retrieve the information needed.

#### Parameters

<a href="#">GIFBElement</a>	Parent class.
-----------------------------	---------------

Definition at line 1568 of file [MemberModel.py](#).

### 7.21.2 Constructor & Destructor Documentation

#### 7.21.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    FibersCircRCCircShape fib,
    int geo_transf_ID,
    lambda_i = -1,
    lambda_j = -1,
    Lp = -1,
    Ip = -1,
    new_integration_ID = -1,
    min_tol = TOL_INTEGRATION,
    max_tol = TOL_INTEGRATION*1e4,
    max_iter = MAX_ITER_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

#### Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fib</i>	(FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).

## Parameters

<i>lambda_i</i>	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
<i>lambda_j</i>	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
<i>Lp</i>	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>min_tol</i>	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>max_tol</i>	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [GIFBElement](#).

Definition at line 1574 of file [MemberModel.py](#).

```

01576         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01577         ele_ID = -1):
01578         """
01579         Constructor of the class.
01580
01581         @param iNode_ID (int): ID of the first end node.
01582         @param jNode_ID (int): ID of the second end node.
01583         @param fib (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
01584         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
01585         documentation).
01586         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
01587         end i (0 = no plastic hinge).
01588         Defaults to -1, e.g. plastic hinge in the end i.
01589         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
01590         end j (0 = no plastic hinge).
01591         Defaults to -1, e.g. plastic hinge in the end j.
01592         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01593         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01594         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01595         e.g. computed in ReInit().
01596         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
01597         to TOL_INTEGRATION (Units).
01598         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
01599         to TOL_INTEGRATION*1e4.
01600         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
01601         convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01602         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01603         IDConvention to define it.
01604         """
01605         self.section = deepcopy(fib.section)
01606         super().__init__(iNode_ID, jNode_ID, fib.ID, self.section.D_bars, self.section.fy,
01607         geo_transf_ID,
01608         lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01609         min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01610         self.section_name_tag = self.section.name_tag
01611         self.UpdateStoredData()
01612         # Check length
01613         self._CheckL()
01614
01615
01616
01617

```

### 7.21.3 Member Data Documentation

### 7.21.3.1 section

section

Definition at line 1596 of file [MemberModel.py](#).

### 7.21.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1600 of file [MemberModel.py](#).

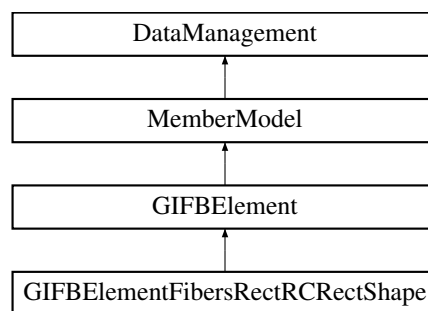
The documentation for this class was generated from the following file:

- /media/carmin/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.22 GIFBElementFibersRectRCRectShape Class Reference

Class that is the children of [GIFBElement](#) and combine the class [FibersRectRCRectShape](#) (fiber section) to retrieve the information needed.

Inheritance diagram for GIFBElementFibersRectRCRectShape:



### Public Member Functions

- def `__init__` (self, int iNode\_ID, int jNode\_ID, [FibersRectRCRectShape](#) fib, int [geo\\_transf\\_ID](#), [lambda\\_i](#)=1, [lambda\\_j](#)=-1, [Lp](#)=-1, [lp](#)=-1, [new\\_integration\\_ID](#)=-1, [min\\_tol](#)=TOL\_INTEGRATION, [max\\_tol](#)=TOL\_INTEGRATION \*1e4, [max\\_iter](#)=MAX\_ITER\_INTEGRATION, ele\_ID=-1)

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.22.1 Detailed Description

Class that is the children of [GIFBElement](#) and combine the class [FibersRectRCRectShape](#) (fiber section) to retrieve the information needed.



## Parameters

<a href="#">GIFBElement</a>	Parent class.
-----------------------------	---------------

Definition at line 1491 of file [MemberModel.py](#).

## 7.22.2 Constructor & Destructor Documentation

### 7.22.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    FibersRectRCRectShape fib,
    int geo_transf_ID,
    lambda_i = -1,
    lambda_j = -1,
    Lp = -1,
    Ip = -1,
    new_integration_ID = -1,
    min_tol = TOL_INTEGRATION,
    max_tol = TOL_INTEGRATION*1e4,
    max_iter = MAX_ITER_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fib</i>	(FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>lambda_i</i>	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
<i>lambda_j</i>	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
<i>Lp</i>	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">ReInit()</a> .
<i>min_tol</i>	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>max_tol</i>	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [GIFBElement](#).

Definition at line 1497 of file [MemberModel.py](#).

```

01499         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
        ele_ID = -1):
01500         """
01501         Constructor of the class.
01502
01503         @param iNode_ID (int): ID of the first end node.
01504         @param jNode_ID (int): ID of the second end node.
01505         @param fib (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
01506         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
01507         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01508             Defaults to -1, e.g. plastic hinge in the end i.
01509         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01510             Defaults to -1, e.g. plastic hinge in the end j.
01511         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01512         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01513         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01514         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION (Units).
01515         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION*1e4.
01516         @param max_iter (int, optional): Maximal number of iteration to reach the integration
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01517         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01518         """
01519         self.section = deepcopy(fib.section)
01520         super().__init__(iNode_ID, jNode_ID, fib.ID, self.section.D_bars, self.section.fy,
geo_transf_ID,
01521             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01522             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01523         self.section_name_tag = self.section.name_tag
01524         self.UpdateStoredData()
01525         # Check length
01526         self._CheckL()
01527
01528

```

## 7.22.3 Member Data Documentation

### 7.22.3.1 section

section

Definition at line 1519 of file [MemberModel.py](#).

### 7.22.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1523 of file [MemberModel.py](#).

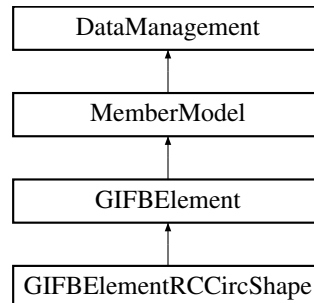
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.23 GIFBElementRCCircShape Class Reference

Class that is the children of [GIFBElement](#) and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for GIFBElementRCCircShape:



### Public Member Functions

- `def __init__(self, int iNode_ID, int jNode_ID, int fiber_ID, RCCircShape section, int geo_transf_ID, lambda_j=-1, lambda_j=-1, Lp=-1, lp=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL_INTEGRATION * 1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.23.1 Detailed Description

Class that is the children of [GIFBElement](#) and combine the class RCCircShape (section) to retrieve the information needed.

#### Parameters

<a href="#">GIFBElement</a>	Parent class.
-----------------------------	---------------

Definition at line 1529 of file [MemberModel.py](#).

#### 7.23.2 Constructor & Destructor Documentation

### 7.23.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    int fiber_ID,
    RCCircShape section,
    int geo_transf_ID,
    lambda_i = -1,
    lambda_j = -1,
    Lp = -1,
    Ip = -1,
    new_integration_ID = -1,
    min_tol = TOL_INTEGRATION,
    max_tol = TOL_INTEGRATION*1e4,
    max_iter = MAX_ITER_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

#### Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber_ID</i>	(int): ID of the fiber section.
<i>section</i>	(RCCircShape): RCCircShape section object.
<i>geo_transf_ID</i>	(int): The geometric transformation (for more information, see OpenSeesPy documentation).
<i>lambda_i</i>	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
<i>lambda_j</i>	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
<i>Lp</i>	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in <a href="#">Relnit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">Relnit()</a> .
<i>min_tol</i>	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>max_tol</i>	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [GIFBElement](#).

Definition at line 1535 of file [MemberModel.py](#).

```
01537     min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
        ele_ID = -1):
01538     """
01539     Constructor of the class.
01540
01541     @param iNode_ID (int): ID of the first end node.
01542     @param jNode_ID (int): ID of the second end node.
01543     @param fiber_ID (int): ID of the fiber section.
01544     @param section (RCCircShape): RCCircShape section object.
```

```

01545         @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
01546         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01547             Defaults to -1, e.g. plastic hinge in the end i.
01548         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01549             Defaults to -1, e.g. plastic hinge in the end j.
01550         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01551         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01552         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01553         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION (Units).
01554         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION*1e4.
01555         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01556         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01557         """
01558         self.section = deepcopy(section)
01559         super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
01560             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01561             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01562         self.section_name_tag = section.name_tag
01563         self.UpdateStoredData()
01564         # Check length
01565         self._CheckL()
01566
01567

```

## 7.23.3 Member Data Documentation

### 7.23.3.1 section

section

Definition at line 1558 of file [MemberModel.py](#).

### 7.23.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1562 of file [MemberModel.py](#).

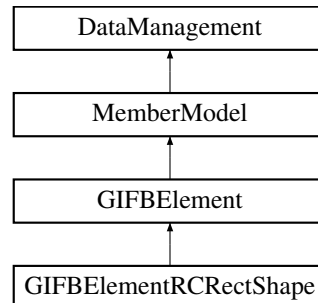
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.24 GIFBElementRRectShape Class Reference

Class that is the children of [GIFBElement](#) and combine the class RRectShape (section) to retrieve the information needed.

Inheritance diagram for GIFBElementRRectShape:



### Public Member Functions

- `def __init__(self, int iNode_ID, int jNode_ID, int fiber_ID, RRectShape section, int geo_transf_ID, lambda_j=-1, lambda_j=-1, Lp=-1, lp=-1, new_integration_ID=-1, min_tol=TOL_INTEGRATION, max_tol=TOL_INTEGRATION * 1e4, max_iter=MAX_ITER_INTEGRATION, ele_ID=-1)`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.24.1 Detailed Description

Class that is the children of [GIFBElement](#) and combine the class RRectShape (section) to retrieve the information needed.

##### Parameters

<a href="#">GIFBElement</a>	Parent class.
-----------------------------	---------------

Definition at line 1452 of file [MemberModel.py](#).

#### 7.24.2 Constructor & Destructor Documentation

7.24.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    int fiber_ID,
    RCRRectShape section,
    int geo_transf_ID,
    lambda_i = -1,
    lambda_j = -1,
    Lp = -1,
    Ip = -1,
    new_integration_ID = -1,
    min_tol = TOL_INTEGRATION,
    max_tol = TOL_INTEGRATION*1e4,
    max_iter = MAX_ITER_INTEGRATION,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>fiber_ID</i>	(int): ID of the fiber section.
<i>section</i>	(RCRectShape): RCRRectShape section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>lambda_i</i>	(float, optional): Fraction of beam length over the plastic hinge length at end i (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end i.
<i>lambda_j</i>	(float, optional): Fraction of beam length over the plastic hinge length at end j (0 = no plastic hinge). Defaults to -1, e.g. plastic hinge in the end j.
<i>Lp</i>	(float, optional): Plastic hinge length. Defaults to -1, e.g. computed in <a href="#">Relnit()</a> .
<i>Ip</i>	(int, optional): Number of integration points (min. 3). Defaults to 5.
<i>new_integration_ID</i>	(int, optional): ID of the integration technique. Defaults to -1, e.g. computed in <a href="#">Relnit()</a> .
<i>min_tol</i>	(float, optional): Minimal tolerance for the integration convergence. Defaults to TOL_INTEGRATION ( <a href="#">Units</a> ).
<i>max_tol</i>	(float, optional): Maximal tolerance for the integration convergence. Defaults to TOL_INTEGRATION*1e4.
<i>max_iter</i>	(int, optional): Maximal number of iteration to reach the integretion convergence. Defaults to MAX_ITER_INTEGRATION ( <a href="#">Units</a> ).
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

Reimplemented from [GIFBElement](#).

Definition at line 1458 of file [MemberModel.py](#).

```
01460     min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
        ele_ID = -1):
01461     """
01462     Constructor of the class.
01463
01464     @param iNode_ID (int): ID of the first end node.
01465     @param jNode_ID (int): ID of the second end node.
01466     @param fiber_ID (int): ID of the fiber section.
01467     @param section (RCRectShape): RCRRectShape section object.
```

```

01468         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
01469         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01470         Defaults to -1, e.g. plastic hinge in the end i.
01471         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01472         Defaults to -1, e.g. plastic hinge in the end j.
01473         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01474         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01475         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01476         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION (Units).
01477         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION*1e4.
01478         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01479         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01480         """
01481         self.section = deepcopy(section)
01482         super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
01483             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01484             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01485         self.section_name_tag = section.name_tag
01486         self.UpdateStoredData()
01487         # Check length
01488         self._CheckL()
01489
01490

```

## 7.24.3 Member Data Documentation

### 7.24.3.1 section

section

Definition at line 1481 of file [MemberModel.py](#).

### 7.24.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1485 of file [MemberModel.py](#).

The documentation for this class was generated from the following file:

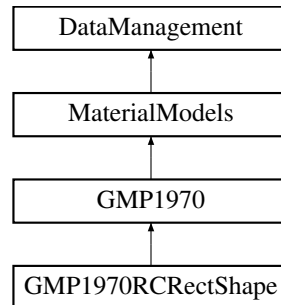
- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)



## 7.25 GMP1970 Class Reference

Class that stores functions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.

Inheritance diagram for GMP1970:



### Public Member Functions

- `def __init__ (self, int ID, fy, Ey, b=0.02, R0=20, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0)`  
*Constructor of the class.*
- `def CheckApplicability (self)`  
*Implementation of the homonym abstract method.*
- `def Relnit (self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self)`  
*Implementation of the homonym abstract method.*
- `def Steel02 (self)`  
*Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

### Public Attributes

- `a1`
- `a2`
- `a3`
- `a4`
- `b`
- `cR1`
- `cR2`
- `data`
- `Ey`
- `fy`
- `ID`
- `Initialized`
- `R0`
- `section_name_tag`

### 7.25.1 Detailed Description

Class that stores functions and material properties of the vertical steel reinforcement bars with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.

For more information about the empirical model for the computation of the parameters, see Giuffré, Menegotto and Pinto 1970 and Carreno et Al. 2020.

## Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 2470 of file [MaterialModels.py](#).

## 7.25.2 Constructor & Destructor Documentation

### 7.25.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int ID,
    fy,
    Ey,
    b = 0.02,
    R0 = 20,
    cR1 = 0.9,
    cR2 = 0.08,
    a1 = 0.039,
    a2 = 1.0,
    a3 = 0.029,
    a4 = 1.0 )
```

Constructor of the class.

The parameters are suggested as exposed in Carreno et Al. 2020 but also the one suggested by OpenSeesPy documentation are reliable ( $b = 0.015$ ,  $R0 = 10$ ,  $cR1 = 0.925$ ,  $cR2 = 0.15$ ).

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>fy</i>	(float): Steel yield strength.
<i>Ey</i>	(float): Young modulus.
<i>b</i>	(float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
<i>R0</i>	(int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.
<i>cR1</i>	(float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
<i>cR2</i>	(float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
<i>a1</i>	(float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.039, according to Carreno et Al. 2020.
<i>a2</i>	(float, optional): Coupled with <i>a1</i> . Defaults to 1.0, according to Carreno et Al. 2020.
<i>a3</i>	(float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.029, according to Carreno et Al. 2020.
<i>a4</i>	(float, optional): Coupled with <i>a3</i> . Defaults to 1.0, according to Carreno et Al. 2020.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
----------------------	------------------------------------

Reimplemented in [GMP1970RCRectShape](#).

Definition at line 2478 of file [MaterialModels.py](#).

```

02478     def __init__(self, ID: int, fy, Ey, b = 0.02, R0 = 20, cR1 = 0.9, cR2 = 0.08, a1 = 0.039, a2 =
1.0, a3 = 0.029, a4 = 1.0):
02479         """
02480         Constructor of the class. The parameters are suggested as exposed in Carreno et Al. 2020 but
also the one suggested by OpenSeesPy documentation are reliable
02481             (b = 0.015, R0 = 10, cR1 = 0.925, cR2 = 0.15).
02482
02483             @param ID (int): Unique material model ID.
02484             @param fy (float): Steel yield strength.
02485             @param Ey (float): Young modulus.
02486             @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et
Al. 2020.
02487             @param R0 (int, optional): First parameter to control the transition from elastic to plastic
branches. Defaults to 20, according to Carreno et Al. 2020.
02488             @param cR1 (float, optional): Second parameter to control the transition from elastic to
plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
02489             @param cR2 (float, optional): Third parameter to control the transition from elastic to
plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
02490             @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield
envelope as proportion of yield strength after a plastic strain.
02491             Defaults to 0.039, according to Carreno et Al. 2020.
02492             @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al.
2020.
02493             @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope
as proportion of yield strength after a plastic strain.
02494             Defaults to 0.029, according to Carreno et Al. 2020.
02495             @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al.
2020.
02496
02497             @exception NegativeValue: ID needs to be a positive integer.
02498             """
02499             # Check
02500             if ID < 1: raise NegativeValue()
02501
02502             # Arguments
02503             self.ID = ID
02504             self.fy = fy
02505             self.Ey = Ey
02506             self.b = b
02507             self.R0 = R0
02508             self.cR1 = cR1
02509             self.cR2 = cR2
02510             self.a1 = a1
02511             self.a2 = a2
02512             self.a3 = a3
02513             self.a4 = a4
02514
02515             # Initialized the parameters that are dependent from others
02516             self.section_name_tag = "None"
02517             self.Initialized = False
02518             self.ReInit()
02519

```

## 7.25.3 Member Function Documentation

### 7.25.3.1 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 2575 of file [MaterialModels.py](#).

```
02575     def CheckApplicability(self):
02576         """
02577         Implementation of the homonym abstract method.
02578         See parent class MaterialModels for detailed information.
02579         """
02580         Check = True
02581         # No checks
02582         if not Check:
02583             print("The validity of the equations is not fullfilled.")
02584             print("!!!!!! WARNING !!!!!!! Check material model of GMP1970, ID=", self.ID)
02585             print("")
02586
02587
```

### 7.25.3.2 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 2520 of file [MaterialModels.py](#).

```
02520     def ReInit(self):
02521         """
02522         Implementation of the homonym abstract method.
02523         See parent class DataManagement for detailed information.
02524         """
02525         # Check applicability
02526         self.CheckApplicability()
02527
02528         # Members
02529         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
02530
02531         # Data storage for loading/saving
02532         self.UpdateStoredData()
02533
02534
```

### 7.25.3.3 ShowInfo()

```
def ShowInfo (
    self )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 2557 of file [MaterialModels.py](#).

```
02557     def ShowInfo(self):
02558         """
02559         Implementation of the homonym abstract method.
02560         See parent class DataManagement for detailed information.
02561         """
02562         print("")
02563         print("Requested info for GMP1970 (Giuffr -Menegotto-Pinto) material model Parameters, ID =
{}".format(self.ID))
```

```

02564         print("Section associated: {} ".format(self.section_name_tag))
02565         print("Yield stress fy = {} MPa".format(self.fy/MPa_unit))
02566         print("Young modulus Ey = {} MPa".format(self.Ey/MPa_unit))
02567         print("Strain hardening ratio b = {}".format(self.b))
02568         print("Bauschinger effect factors R0 = {}, cR1 = {} and cR2 = {}".format(self.R0, self.cR1,
self.cR2))
02569         print("Isotropic hardening factors a1 = {}, a2 = {}, a3 = {} and a4 = {}".format(self.a1,
self.a2, self.a3, self.a4))
02570         print("")
02571
02572         #TODO: add plot option (difficult to implement)
02573
02574

```

### 7.25.3.4 Steel02()

```

def Steel02 (
    self )

```

Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with isotropic strain hardening.

See `_Steel02` function for more information.

Definition at line 2588 of file [MaterialModels.py](#).

```

02588     def Steel02(self):
02589         """
02590         Generate the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel material with
isotropic strain hardening.
02591         See _Steel02 function for more information.
02592         """
02593         _Steel02(self.ID, self.fy, self.Ey, self.b, self.R0, self.cR1, self.cR2, self.a1, self.a2,
self.a3, self.a4)
02594         self.Initialized = True
02595         self.UpdateStoredData()
02596
02597

```

### 7.25.3.5 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 2536 of file [MaterialModels.py](#).

```

02536     def UpdateStoredData(self):
02537         """
02538         Implementation of the homonym abstract method.
02539         See parent class DataManagement for detailed information.
02540         """
02541         self.data = [{"INFO_TYPE", "GMP1970"}, # Tag for differentiating different data
02542                     ["ID", self.ID],
02543                     ["section_name_tag", self.section_name_tag],
02544                     ["fy", self.fy],
02545                     ["Ey", self.Ey],
02546                     ["b", self.b],
02547                     ["R0", self.R0],
02548                     ["cR1", self.cR1],
02549                     ["cR2", self.cR2],
02550                     ["a1", self.a1],
02551                     ["a2", self.a2],
02552                     ["a3", self.a3],
02553                     ["a4", self.a4],
02554                     ["Initialized", self.Initialized]]
02555
02556

```

## 7.25.4 Member Data Documentation

### 7.25.4.1 a1

a1

Definition at line 2510 of file [MaterialModels.py](#).

### 7.25.4.2 a2

a2

Definition at line 2511 of file [MaterialModels.py](#).

### 7.25.4.3 a3

a3

Definition at line 2512 of file [MaterialModels.py](#).

### 7.25.4.4 a4

a4

Definition at line 2513 of file [MaterialModels.py](#).

### 7.25.4.5 b

b

Definition at line 2506 of file [MaterialModels.py](#).

**7.25.4.6 cR1**

cR1

Definition at line [2508](#) of file [MaterialModels.py](#).

**7.25.4.7 cR2**

cR2

Definition at line [2509](#) of file [MaterialModels.py](#).

**7.25.4.8 data**

data

Definition at line [2541](#) of file [MaterialModels.py](#).

**7.25.4.9 Ey**

Ey

Definition at line [2505](#) of file [MaterialModels.py](#).

**7.25.4.10 fy**

fy

Definition at line [2504](#) of file [MaterialModels.py](#).

**7.25.4.11 ID**

ID

Definition at line [2503](#) of file [MaterialModels.py](#).



#### 7.25.4.12 Initialized

Initialized

Definition at line 2517 of file [MaterialModels.py](#).

#### 7.25.4.13 R0

R0

Definition at line 2507 of file [MaterialModels.py](#).

#### 7.25.4.14 section\_name\_tag

section\_name\_tag

Definition at line 2516 of file [MaterialModels.py](#).

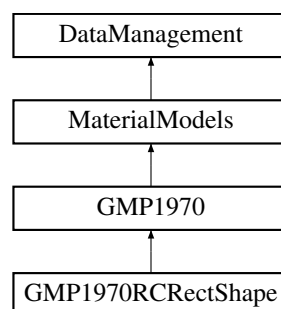
The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[MaterialModels.py](#)

## 7.26 GMP1970RCRectShape Class Reference

Class that is the children of [GMP1970](#) and combine the class RRectShape (section) to retrieve the information needed.

Inheritance diagram for GMP1970RCRectShape:



### Public Member Functions

- `def __init__(self, int ID, RRectShape section, b=0.02, R0=20.0, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0)`  
*Constructor of the class.*

## Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.26.1 Detailed Description

Class that is the children of [GMP1970](#) and combine the class `RRectShape` (`section`) to retrieve the information needed.

#### Parameters

<a href="#">GMP1970</a>	Parent class.
-------------------------	---------------

Definition at line [2598](#) of file [MaterialModels.py](#).

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RRectShape section,
    b = 0.02,
    R0 = 20.0,
    cR1 = 0.9,
    cR2 = 0.08,
    a1 = 0.039,
    a2 = 1.0,
    a3 = 0.029,
    a4 = 1.0 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class `RRectShape`. The copy of the section passed is stored in the member variable `self.section`.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	( <code>RRectShape</code> ): <code>RRectShape</code> section object.
<i>b</i>	(float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et Al. 2020.
<i>R0</i>	(int, optional): First parameter to control the transition from elastic to plastic branches. Defaults to 20, according to Carreno et Al. 2020.

## Parameters

<i>cR1</i>	(float, optional): Second parameter to control the transition from elastic to plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
<i>cR2</i>	(float, optional): Third parameter to control the transition from elastic to plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
<i>a1</i>	(float, optional): Isotropic hardening parameter, increase of compression yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.039, according to Carreno et Al. 2020.
<i>a2</i>	(float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al. 2020.
<i>a3</i>	(float, optional): Isotropic hardening parameter, increase of tension yield envelope as proportion of yield strength after a plastic strain. Defaults to 0.029, according to Carreno et Al. 2020.
<i>a4</i>	(float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al. 2020.

Reimplemented from [GMP1970](#).

Definition at line 2604 of file [MaterialModels.py](#).

```

02604     def __init__(self, ID: int, section: RCRectShape, b=0.02, R0=20.0, cR1=0.9, cR2=0.08, a1=0.039,
02605                 a2=1.0, a3=0.029, a4=1.0):
02606         """
02607         Constructor of the class. It passes the arguments into the parent class to generate the
02608         combination of the parent class
02609         and the section class RCRectShape.
02610         The copy of the section passed is stored in the member variable self.section.
02611         @param ID (int): Unique material model ID.
02612         @param section (RCRectShape): RCRectShape section object.
02613         @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et
02614         Al. 2020.
02615         @param R0 (int, optional): First parameter to control the transition from elastic to plastic
02616         branches. Defaults to 20, according to Carreno et Al. 2020.
02617         @param cR1 (float, optional): Second parameter to control the transition from elastic to
02618         plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
02619         @param cR2 (float, optional): Third parameter to control the transition from elastic to
02620         plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
02621         @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield
02622         envelope as proportion of yield strength after a plastic strain.
02623         Defaults to 0.039, according to Carreno et Al. 2020.
02624         @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al.
02625         2020.
02626         @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope
02627         as proportion of yield strength after a plastic strain.
02628         Defaults to 0.029, according to Carreno et Al. 2020.
02629         @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al.
02630         2020.
02631         """
02632         self.section = deepcopy(section)
02633         super().__init__(ID, section.fy, section.Ey, b=b, R0=R0, cR1=cR1, cR2=cR2, a1=a1, a2=a2,
02634                         a3=a3, a4=a4)
02635         self.section_name_tag = section.name_tag
02636         self.UpdateStoredData()
02637
02638

```

## 7.26.3 Member Data Documentation

### 7.26.3.1 section

section

Definition at line 2623 of file [MaterialModels.py](#).

### 7.26.3.2 section\_name\_tag

section\_name\_tag

Definition at line 2625 of file [MaterialModels.py](#).

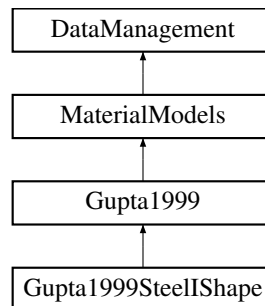
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.27 Gupta1999 Class Reference

Class that stores functions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

Inheritance diagram for Gupta1999:



### Public Member Functions

- `def __init__ (self, int ID, d_c, bf_c, tf_c, l_c, d_b, tf_b, Fy, E, t_p, t_dp=0.0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0.0, dmg2=0.0, beta=0.0, safety_factor=False)`  
*Constructor of the class.*
- `def CheckApplicability (self)`  
*Implementation of the homonym abstract method.*
- `def Hysteretic (self)`  
*Generate the material model Hysteretic (Gupta 1999) using the computed parameters.*
- `def Relnit (self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

## Public Attributes

- [a\\_s](#)
- [beam\\_section\\_name\\_tag](#)
- [beta](#)
- [bf\\_c](#)
- [col\\_section\\_name\\_tag](#)
- [d\\_b](#)
- [d\\_c](#)
- [data](#)
- [dmg1](#)
- [dmg2](#)
- [E](#)
- [Fy](#)
- [G](#)
- [gamma1\\_y](#)
- [gamma2\\_y](#)
- [gamma3\\_y](#)
- [I\\_c](#)
- [ID](#)
- [Initialized](#)
- [Ke](#)
- [Kp](#)
- [M1y](#)
- [M2y](#)
- [M3y](#)
- [pinchx](#)
- [pinchy](#)
- [Ry](#)
- [t\\_dp](#)
- [t\\_p](#)
- [t\\_pz](#)
- [tf\\_b](#)
- [tf\\_c](#)
- [Vy](#)

### 7.27.1 Detailed Description

Class that stores functions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

The material model is valid only if the column is continuous. For more information about the empirical model for the computation of the parameters, see Gupta 1999.

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 522 of file [MaterialModels.py](#).

## 7.27.2 Constructor & Destructor Documentation

### 7.27.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    float d_c,
    float bf_c,
    float tf_c,
    float I_c,
    float d_b,
    float tf_b,
    float Fy,
    float E,
    float t_p,
    float t_dp = 0.0,
    float a_s = 0.03,
    float pinchx = 0.25,
    float pinchy = 0.75,
    float dmg1 = 0.0,
    float dmg2 = 0.0,
    float beta = 0.0,
    bool safety_factor = False )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>d_c</i>	(float): Column depth.
<i>bf_c</i>	(float): Column flange width.
<i>tf_c</i>	(float): Column flange thickness.
<i>I_c</i>	(float): Column moment of inertia (strong axis).
<i>d_b</i>	(float): Beam depth.
<i>tf_b</i>	(float): Beam flange thickness.
<i>Fy</i>	(float): Yield strength (if assume continous column, Fy of the web).
<i>E</i>	(float): Young modulus.
<i>t_p</i>	(float): Panel zone thickness.
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.0.
<i>a_s</i>	(float, optional): Strain hardening. Defaults to 0.03.
<i>pinchx</i>	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
<i>pinchy</i>	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
<i>dmg1</i>	(float, optional): Damage due to ductility: D1( $\mu$ -1). Defaults to 0.0.
<i>dmg2</i>	(float, optional): Damage due to energy: D2( $E_{ii}/E_{ult}$ ). Defaults to 0.0.
<i>beta</i>	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, $\mu$ -beta. Defaults to 0.0.
<i>safety_factor</i>	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	d_c needs to be positive.
<i>NegativeValue</i>	bf_c needs to be positive.
<i>NegativeValue</i>	tf_c needs to be positive.
<i>NegativeValue</i>	d_b needs to be positive.
<i>NegativeValue</i>	tf_b needs to be positive.
<i>NegativeValue</i>	Fy needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	t_p needs to be positive.
<i>NegativeValue</i>	a_s needs to be positive.

Reimplemented in [Gupta1999SteelShape](#).

Definition at line 531 of file [MaterialModels.py](#).

```

00532         t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
        safety_factor = False):
00533     """
00534     Constructor of the class.
00535
00536     @param ID (int): Unique material model ID.
00537     @param d_c (float): Column depth.
00538     @param bf_c (float): Column flange width.
00539     @param tf_c (float): Column flange thickness.
00540     @param I_c (float): Column moment of inertia (strong axis).
00541     @param d_b (float): Beam depth.
00542     @param tf_b (float): Beam flange thickness.
00543     @param Fy (float): Yield strength (if assume continous column, Fy of the web).
00544     @param E (float): Young modulus.
00545     @param t_p (float): Panel zone thickness.
00546     @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00547     @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00548     @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
        Defaults to 0.25.
00549     @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
        Defaults to 0.75.
00550     @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00551     @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00552     @param beta (float, optional): Power used to determine the degraded unloading stiffness based
        on ductility, mu-beta. Defaults to 0.0.
00553     @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
        are used (not test results). Defaults to False.
00554
00555     @exception NegativeValue: ID needs to be a positive integer.
00556     @exception NegativeValue: d_c needs to be positive.
00557     @exception NegativeValue: bf_c needs to be positive.
00558     @exception NegativeValue: tf_c needs to be positive.
00559     @exception NegativeValue: d_b needs to be positive.
00560     @exception NegativeValue: tf_b needs to be positive.
00561     @exception NegativeValue: Fy needs to be positive.
00562     @exception NegativeValue: E needs to be positive.
00563     @exception NegativeValue: t_p needs to be positive.
00564     @exception NegativeValue: a_s needs to be positive.
00565     """
00566     # Check
00567     if ID < 1: raise NegativeValue()
00568     if d_c < 0: raise NegativeValue()
00569     if bf_c < 0: raise NegativeValue()
00570     if tf_c < 0: raise NegativeValue()
00571     if d_b < 0: raise NegativeValue()
00572     if tf_b < 0: raise NegativeValue()
00573     if Fy < 0: raise NegativeValue()
00574     if E < 0: raise NegativeValue()
00575     if t_p < 0: raise NegativeValue()
00576     if a_s < 0: raise NegativeValue()
00577
00578     # Arguments
00579     self.ID = ID
00580     self.d_c = d_c
00581     self.bf_c = bf_c
00582     self.tf_c = tf_c
00583     self.I_c = I_c
00584     self.d_b = d_b
00585     self.tf_b = tf_b

```

```

00586         self.Fy = Fy
00587         self.E = E
00588         self.t_p = t_p
00589         self.t_dp = t_dp
00590         self.a_s = a_s
00591         self.pinchx = pinchx
00592         self.pinchy = pinchy
00593         self.dmg1 = dmg1
00594         self.dmg2 = dmg2
00595         self.beta = beta
00596         if safety_factor:
00597             self.Ry = 1.2
00598         else:
00599             self.Ry = 1.0
00600
00601         # Initialized the parameters that are dependent from others
00602         self.beam_section_name_tag = "None"
00603         self.col_section_name_tag = "None"
00604         self.Initialized = False
00605         self.ReInit()
00606
00607

```

## 7.27.3 Member Function Documentation

### 7.27.3.1 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 722 of file [MaterialModels.py](#).

```

00722     def CheckApplicability(self):
00723         """
00724         Implementation of the homonym abstract method.
00725         See parent class MaterialModels for detailed information.
00726         """
00727         Check = True
00728         # No checks
00729         if not Check:
00730             print("The validity of the equations is not fullfilled.")
00731             print("!!!!!! WARNING !!!!!!! Check material model of Gupta 1999, ID=", self.ID)
00732             print("")
00733
00734

```

### 7.27.3.2 Hysteretic()

```

def Hysteretic (
    self )

```

Generate the material model Hysteretic (Gupta 1999) using the computed parameters.

See `_Hysteretic` function for more information.

Definition at line 735 of file [MaterialModels.py](#).



```

00735     def Hysteretic(self):
00736         """
00737         Generate the material model Hysteretic (Gupta 1999) using the computed parameters.
00738         See _Hysteretic function for more information.
00739         """
00740         _Hysteretic(self.ID, self.M1y, self.gammal_y, self.M2y, self.gamma2_y, self.M3y,
self.gamma3_y,
00741                     self.pinchx, self.pinchy, self.dmg1, self.dmg2, self.beta)
00742         self.Initialized = True
00743         self.UpdateStoredData()
00744
00745

```

### 7.27.3.3 ReInit()

```

def ReInit (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 609 of file [MaterialModels.py](#).

```

00609     def ReInit(self):
00610         """
00611         Implementation of the homonym abstract method.
00612         See parent class DataManagement for detailed information.
00613         """
00614         # Check applicability
00615         self.CheckApplicability()
00616
00617         # Members
00618         if self.beam_section_name_tag != "None": self.beam_section_name_tag =
self.beam_section_name_tag + " (modified)"
00619         if self.col_section_name_tag != "None": self.col_section_name_tag = self.col_section_name_tag
+ " (modified)"
00620
00621         # Trilinear Parameters
00622         self.t_pz = self.t_p + self.t_dp
00623         self.Vy = 0.55 * self.Fy * self.Ry * self.d_c * self.t_pz # Yield Shear
00624         self.G = self.E/(2.0 * (1.0 + 0.30)) # Shear Modulus
00625         self.Ke = 0.95 * self.G * self.t_pz * self.d_c # Elastic Stiffness
00626         self.Kp = 0.95 * self.G * self.bf_c * (self.tf_c * self.tf_c) / self.d_b # Plastic Stiffness
00627
00628         # Define Trilinear Equivalent Rotational Spring
00629         # Yield point for Trilinear Spring at gammal_y
00630         self.gammal_y = self.Vy / self.Ke
00631         self.M1y = self.gammal_y * (self.Ke * self.d_b)
00632         # Second Point for Trilinear Spring at 4 * gammal_y
00633         self.gamma2_y = 4.0 * self.gammal_y
00634         self.M2y = self.M1y + (self.Kp * self.d_b) * (self.gamma2_y - self.gammal_y)
00635         # Third Point for Trilinear Spring at 100 * gammal_y
00636         self.gamma3_y = 100.0 * self.gammal_y
00637         self.M3y = self.M2y + (self.a_s * self.Ke * self.d_b) * (self.gamma3_y - self.gamma2_y)
00638
00639         # Data storage for loading/saving
00640         self.UpdateStoredData()
00641
00642

```

### 7.27.3.4 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

## Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 682 of file [MaterialModels.py](#).

```

00682     def ShowInfo(self, plot = False, block = False):
00683         """
00684         Implementation of the homonym abstract method.
00685         See parent class DataManagement for detailed information.
00686
00687         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00688         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00689         """
00690         print("")
00691         print("Requested info for Gupta 1999 material model Parameters, ID = {}".format(self.ID))
00692         print("Sections associated, column: {}".format(self.col_section_name_tag))
00693         print("Sections associated, beam: {}".format(self.beam_section_name_tag))
00694         print("gamma1_y = {} rad".format(self.gamma1_y))
00695         print("gamma2_y = {} rad".format(self.gamma2_y))
00696         print("gamma3_y = {} rad".format(self.gamma3_y))
00697         print("M1y = {} kNm".format(self.M1y/kNm_unit))
00698         print("M2y = {} kNm".format(self.M2y/kNm_unit))
00699         print("M3y = {} kNm".format(self.M3y/kNm_unit))
00700         print("")
00701
00702         if plot:
00703             # Data for plotting
00704             # Last point for plot
00705             gamma3_y_plot = 10.0 * self.gamma1_y
00706             M3y_plot = self.M2y + (self.a_s * self.Ke * self.d_b) * (gamma3_y_plot - self.gamma2_y)
00707
00708             x_axis = np.array([0.0, self.gamma1_y, self.gamma2_y, gamma3_y_plot])
00709             y_axis = np.array([0.0, self.M1y, self.M2y, M3y_plot])/kNm_unit
00710
00711             fig, ax = plt.subplots()
00712             ax.plot(x_axis, y_axis, 'k-')
00713
00714             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
00715                   title='Gupta 1999 material model (ID={})'.format(self.ID))
00716             ax.grid()
00717
00718             if block:
00719                 plt.show()
00720
00721

```

### 7.27.3.5 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 643 of file [MaterialModels.py](#).

```

00643     def UpdateStoredData(self):
00644         """
00645         Implementation of the homonym abstract method.
00646         See parent class DataManagement for detailed information.
00647         """
00648         self.data = [{"INFO_TYPE", "Gupta1999"}, # Tag for differentiating different data
00649                      {"ID", self.ID},
00650                      {"beam_section_name_tag", self.beam_section_name_tag},
00651                      {"col_section_name_tag", self.col_section_name_tag},
00652                      {"d_c", self.d_c},
00653                      {"bf_c", self.bf_c},

```

```

00654         ["tf_c", self.tf_c],
00655         ["I_c", self.I_c],
00656         ["d_b", self.d_b],
00657         ["tf_b", self.tf_b],
00658         ["Fy", self.Fy],
00659         ["E", self.E],
00660         ["G", self.G],
00661         ["t_p", self.t_p],
00662         ["t_dp", self.t_dp],
00663         ["t_pz", self.t_pz],
00664         ["a_s", self.a_s],
00665         ["pinchx", self.pinchx],
00666         ["pinchy", self.pinchy],
00667         ["dmg1", self.dmg1],
00668         ["dmg2", self.dmg2],
00669         ["beta", self.beta],
00670         ["Ry", self.Ry],
00671         ["Vy", self.Vy],
00672         ["Ke", self.Ke],
00673         ["Kp", self.Kp],
00674         ["gamma1_y", self.gamma1_y],
00675         ["M1y", self.M1y],
00676         ["gamma2_y", self.gamma2_y],
00677         ["M2y", self.M2y],
00678         ["gamma3_y", self.gamma3_y],
00679         ["M3y", self.M3y],
00680         ["Initialized", self.Initialized]]
00681

```

## 7.27.4 Member Data Documentation

### 7.27.4.1 a\_s

a\_s

Definition at line 590 of file [MaterialModels.py](#).

### 7.27.4.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 602 of file [MaterialModels.py](#).

### 7.27.4.3 beta

beta

Definition at line 595 of file [MaterialModels.py](#).

#### 7.27.4.4 `bf_c`

`bf_c`

Definition at line 581 of file [MaterialModels.py](#).

#### 7.27.4.5 `col_section_name_tag`

`col_section_name_tag`

Definition at line 603 of file [MaterialModels.py](#).

#### 7.27.4.6 `d_b`

`d_b`

Definition at line 584 of file [MaterialModels.py](#).

#### 7.27.4.7 `d_c`

`d_c`

Definition at line 580 of file [MaterialModels.py](#).

#### 7.27.4.8 `data`

`data`

Definition at line 648 of file [MaterialModels.py](#).

#### 7.27.4.9 `dmg1`

`dmg1`

Definition at line 593 of file [MaterialModels.py](#).

**7.27.4.10 dmg2**

dmg2

Definition at line 594 of file [MaterialModels.py](#).

**7.27.4.11 E**

E

Definition at line 587 of file [MaterialModels.py](#).

**7.27.4.12 Fy**

Fy

Definition at line 586 of file [MaterialModels.py](#).

**7.27.4.13 G**

G

Definition at line 624 of file [MaterialModels.py](#).

**7.27.4.14 gamma1\_y**

gamma1\_y

Definition at line 630 of file [MaterialModels.py](#).

**7.27.4.15 gamma2\_y**

gamma2\_y

Definition at line 633 of file [MaterialModels.py](#).

#### 7.27.4.16 `gamma3_y`

`gamma3_y`

Definition at line 636 of file [MaterialModels.py](#).

#### 7.27.4.17 `I_c`

`I_c`

Definition at line 583 of file [MaterialModels.py](#).

#### 7.27.4.18 `ID`

`ID`

Definition at line 579 of file [MaterialModels.py](#).

#### 7.27.4.19 `Initialized`

`Initialized`

Definition at line 604 of file [MaterialModels.py](#).

#### 7.27.4.20 `Ke`

`Ke`

Definition at line 625 of file [MaterialModels.py](#).

#### 7.27.4.21 `Kp`

`Kp`

Definition at line 626 of file [MaterialModels.py](#).

**7.27.4.22 M1y**

M1y

Definition at line 631 of file [MaterialModels.py](#).

**7.27.4.23 M2y**

M2y

Definition at line 634 of file [MaterialModels.py](#).

**7.27.4.24 M3y**

M3y

Definition at line 637 of file [MaterialModels.py](#).

**7.27.4.25 pinchx**

pinchx

Definition at line 591 of file [MaterialModels.py](#).

**7.27.4.26 pinchy**

pinchy

Definition at line 592 of file [MaterialModels.py](#).

**7.27.4.27 Ry**

Ry

Definition at line 597 of file [MaterialModels.py](#).

**7.27.4.28 t\_dp**

t\_dp

Definition at line 589 of file [MaterialModels.py](#).

**7.27.4.29 t\_p**

t\_p

Definition at line 588 of file [MaterialModels.py](#).

**7.27.4.30 t\_pz**

t\_pz

Definition at line 622 of file [MaterialModels.py](#).

**7.27.4.31 tf\_b**

tf\_b

Definition at line 585 of file [MaterialModels.py](#).

**7.27.4.32 tf\_c**

tf\_c

Definition at line 582 of file [MaterialModels.py](#).

**7.27.4.33 Vy**

Vy

Definition at line 623 of file [MaterialModels.py](#).

The documentation for this class was generated from the following file:

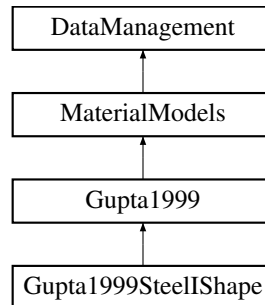
- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)



## 7.28 Gupta1999SteelShape Class Reference

Class that is the children of [Gupta1999](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for Gupta1999SteelShape:



### Public Member Functions

- `def __init__(self, int ID, SteelShape col, SteelShape beam, t_dp=0.0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0.0, dmg2=0.0, beta=0.0, safety_factor=False)`

*Constructor of the class.*

### Public Attributes

- `beam`
- `beam_section_name_tag`
- `col`
- `col_section_name_tag`

#### 7.28.1 Detailed Description

Class that is the children of [Gupta1999](#) and combine the class SteelShape (section) to retrieve the information needed.

#### Parameters

<a href="#">Gupta1999</a>	Parent class.
---------------------------	---------------

Definition at line 746 of file [MaterialModels.py](#).

#### 7.28.2 Constructor & Destructor Documentation

### 7.28.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    SteelIShape col,
    SteelIShape beam,
    t_dp = 0.0,
    a_s = 0.03,
    pinchx = 0.25,
    pinchy = 0.75,
    dmg1 = 0.0,
    dmg2 = 0.0,
    beta = 0.0,
    safety_factor = False )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. The copy of the sections (col and beam) passed is stored in the member variable self.section.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>col</i>	(SteelIShape): SteelIShape column section object.
<i>beam</i>	(SteelIShape): SteelIShape beam section object.
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.0.
<i>a_s</i>	(float, optional): Strain hardening. Defaults to 0.03.
<i>pinchx</i>	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
<i>pinchy</i>	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
<i>dmg1</i>	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
<i>dmg2</i>	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
<i>beta</i>	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
<i>safety_factor</i>	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

Reimplemented from [Gupta1999](#).

Definition at line 752 of file [MaterialModels.py](#).

```
00753     t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
    safety_factor = False):
00754     """
00755     Constructor of the class. It passes the arguments into the parent class to generate the
    combination of the parent class
00756     and the section class SteelIShape.
00757     The copy of the sections (col and beam) passed is stored in the member variable self.section.
00758
00759     @param ID (int): Unique material model ID.
00760     @param col (SteelIShape): SteelIShape column section object.
00761     @param beam (SteelIShape): SteelIShape beam section object.
00762     @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00763     @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00764     @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
    Defaults to 0.25.
00765     @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
    Defaults to 0.75
00766     @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00767     @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00768     @param beta (float, optional): Power used to determine the degraded unloading stiffness based
    on ductility, mu-beta. Defaults to 0.0.
```

```

00769         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
00770         are used (not test results). Defaults to False.
00771         """
00772         self.col = deepcopy(col)
00773         self.beam = deepcopy(beam)
00774         super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E,
00775         col.tw,
00776         t_dp, a_s, pinchx, pinchy, dmg1, dmg2, beta, safety_factor)
00777         self.beam_section_name_tag = beam.name_tag
00778         self.col_section_name_tag = col.name_tag
00779         self.UpdateStoredData()

```

## 7.28.3 Member Data Documentation

### 7.28.3.1 beam

beam

Definition at line 772 of file [MaterialModels.py](#).

### 7.28.3.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 775 of file [MaterialModels.py](#).

### 7.28.3.3 col

col

Definition at line 771 of file [MaterialModels.py](#).

### 7.28.3.4 col\_section\_name\_tag

col\_section\_name\_tag

Definition at line 776 of file [MaterialModels.py](#).

The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[MaterialModels.py](#)

## 7.29 IDGenerator Class Reference

Class that manage the ID generation.

### Public Member Functions

- `def __init__ (self)`  
*The class constructor.*
- `def GenerateIDElement (self)`  
*Method that generate a unique element ID.*
- `def GenerateIDFiber (self)`  
*Method that generate a unique fiber ID.*
- `def GenerateIDMat (self)`  
*Method that generate a unique material ID.*
- `def GenerateIDNode (self)`  
*Method that generate a unique node ID.*

### Public Attributes

- `current_element_ID`
- `current_fiber_ID`
- `current_mat_ID`
- `current_node_ID`

### 7.29.1 Detailed Description

Class that manage the ID generation.

USE ONLY IF EVERY NODE IS DEFINED BY THE USER (because the OpenSeesPyAssistant modules use the convention defined in the functions above).

Definition at line 412 of file [FunctionalFeatures.py](#).

### 7.29.2 Constructor & Destructor Documentation

#### 7.29.2.1 \_\_init\_\_()

```
def __init__ (
    self )
```

The class constructor.

Definition at line 416 of file [FunctionalFeatures.py](#).

```
00416     def __init__(self):
00417         """The class constructor.
00418         """
00419         self.current_node_ID = 0
00420         self.current_element_ID = 0
00421         self.current_mat_ID = 0
00422         self.current_fiber_ID = 0
00423
```

## 7.29.3 Member Function Documentation

### 7.29.3.1 GenerateIDElement()

```
def GenerateIDElement (
    self )
```

Method that generate a unique element ID.

#### Returns

int: The element ID.

Definition at line 433 of file [FunctionalFeatures.py](#).

```
00433     def GenerateIDElement(self):
00434         """
00435         Method that generate a unique element ID.
00436
00437         @returns int: The element ID.
00438         """
00439         self.current_element_ID = self.current_element_ID + 1
00440         return self.current_element_ID
00441
```

### 7.29.3.2 GenerateIDFiber()

```
def GenerateIDFiber (
    self )
```

Method that generate a unique fiber ID.

#### Returns

int: The fiber ID.

Definition at line 451 of file [FunctionalFeatures.py](#).

```
00451     def GenerateIDFiber(self):
00452         """
00453         Method that generate a unique fiber ID.
00454
00455         @returns int: The fiber ID.
00456         """
00457         self.current_fiber_ID = self.current_fiber_ID + 1
00458         return self.current_fiber_ID
00459
00460
```

### 7.29.3.3 GenerateIDMat()

```
def GenerateIDMat (
    self )
```

Method that generate a unique material ID.

#### Returns

int: The material ID.

Definition at line 442 of file [FunctionalFeatures.py](#).

```
00442     def GenerateIDMat(self):
00443         """
00444         Method that generate a unique material ID.
00445
00446         @returns int: The material ID.
00447         """
00448         self.current_mat_ID = self.current_mat_ID + 1
00449         return self.current_mat_ID
00450
```

### 7.29.3.4 GenerateIDNode()

```
def GenerateIDNode (
    self )
```

Method that generate a unique node ID.

#### Returns

int: The node ID.

Definition at line 424 of file [FunctionalFeatures.py](#).

```
00424     def GenerateIDNode(self):
00425         """
00426         Method that generate a unique node ID.
00427
00428         @returns int: The node ID.
00429         """
00430         self.current_node_ID = self.current_node_ID + 1
00431         return self.current_node_ID
00432
```

## 7.29.4 Member Data Documentation

### 7.29.4.1 current\_element\_ID

```
current_element_ID
```

Definition at line 420 of file [FunctionalFeatures.py](#).

#### 7.29.4.2 `current_fiber_ID`

`current_fiber_ID`

Definition at line 422 of file [FunctionalFeatures.py](#).

#### 7.29.4.3 `current_mat_ID`

`current_mat_ID`

Definition at line 421 of file [FunctionalFeatures.py](#).

#### 7.29.4.4 `current_node_ID`

`current_node_ID`

Definition at line 419 of file [FunctionalFeatures.py](#).

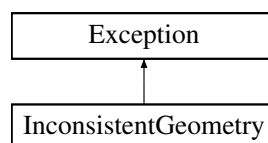
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/FunctionalFeatures.py](#)

## 7.30 InconsistentGeometry Class Reference

Exception class for the "inconsistent geometry" error.

Inheritance diagram for InconsistentGeometry:



### 7.30.1 Detailed Description

Exception class for the "inconsistent geometry" error.

Definition at line 31 of file [ErrorHandling.py](#).

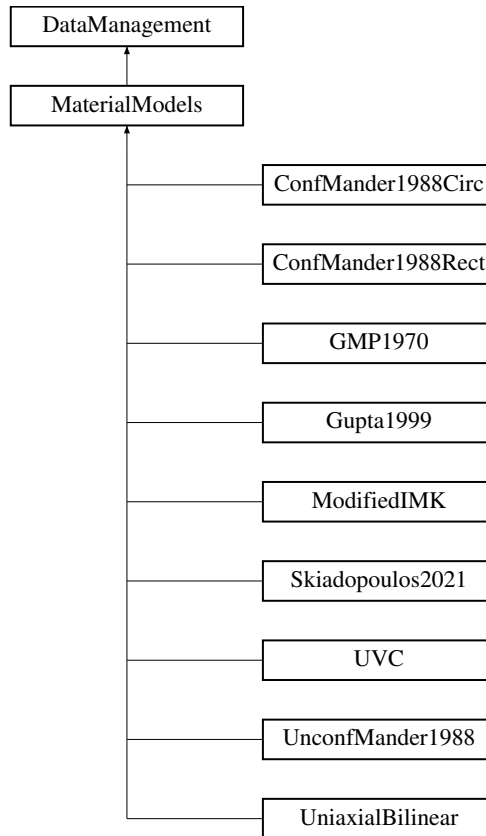
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.31 MaterialModels Class Reference

Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.

Inheritance diagram for MaterialModels:



### Public Member Functions

- def [CheckApplicability](#) (self)

*Abstract function used to check the applicability of the material model.*

#### 7.31.1 Detailed Description

Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.

##### Parameters

<i>DataManagement</i>	Parent abstract class.
-----------------------	------------------------

Definition at line 19 of file [MaterialModels.py](#).



## 7.31.2 Member Function Documentation

### 7.31.2.1 CheckApplicability()

```
def CheckApplicability (
    self )
```

Abstract function used to check the applicability of the material model.

Reimplemented in [ModifiedIMK](#), [Gupta1999](#), [Skiadopoulos2021](#), [UnconfMander1988](#), [ConfMander1988Rect](#), [ConfMander1988Circ](#), [UniaxialBilinear](#), [GMP1970](#), and [UVC](#).

Definition at line 27 of file [MaterialModels.py](#).

```
00027     def CheckApplicability(self):
00028         """
00029         Abstract function used to check the applicability of the material model.
00030         """
00031         pass
00032
00033
```

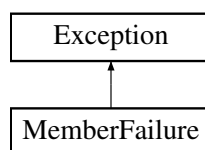
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.32 MemberFailure Class Reference

Exception class for the "member failure" error.

Inheritance diagram for MemberFailure:



### 7.32.1 Detailed Description

Exception class for the "member failure" error.

Definition at line 36 of file [ErrorHandling.py](#).

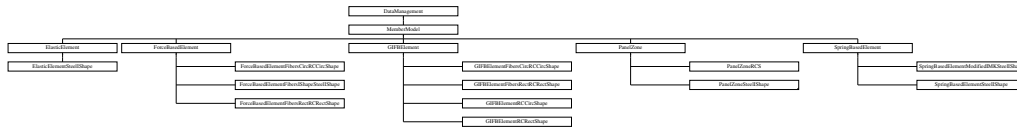
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.33 MemberModel Class Reference

Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for MemberModel:



### Public Member Functions

- def [Record](#) (self, ele\_ID, str name\_txt, str data\_dir, force\_rec=True, def\_rec=True, time\_rec=True)  
Abstract method that records the forces, deformation and time of the member associated with the class.
- def [RecordNodeDef](#) (self, int iNode\_ID, int jNode\_ID, str name\_txt, str data\_dir, time\_rec=True)  
Abstract method that records the deformation and time of the member's nodes associated with the class.

#### 7.33.1 Detailed Description

Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

##### Parameters

<i>DataManagement</i>	Parent abstract class.
-----------------------	------------------------

Definition at line 22 of file [MemberModel.py](#).

#### 7.33.2 Member Function Documentation

##### 7.33.2.1 Record()

```
def Record (
    self,
    ele_ID,
    str name_txt,
    str data_dir,
    force_rec = True,
    def_rec = True,
    time_rec = True )
```

Abstract method that records the forces, deformation and time of the member associated with the class.

## Parameters

<i>ele_ID</i>	(int): The ID of the element that will be recorded.
<i>name_txt</i>	(str): Name of the recorded data (no .txt).
<i>data_dir</i>	(str): Directory for the storage of data.
<i>force_rec</i>	(bool, optional): Option to record the forces (Fx, Fy, Mz). Defaults to True.
<i>def_rec</i>	(bool, optional): Option to record the deformation (theta) for ZeroLength element. Defaults to True.
<i>time_rec</i>	(bool, optional): Option to record time. Defaults to True.

Reimplemented in [PanelZone](#), [ElasticElement](#), [ForceBasedElement](#), [GIFBElement](#), and [SpringBasedElement](#).

Definition at line 29 of file [MemberModel.py](#).

```

00029     def Record(self, ele_ID: str, data_dir: str, force_rec = True, def_rec = True, time_rec
= True):
00030         """
00031         Abstract method that records the forces, deformation and time of the member associated with
the class.
00032
00033         @param ele_ID (int): The ID of the element that will be recorded.
00034         @param name_txt (str): Name of the recorded data (no .txt).
00035         @param data_dir (str): Directory for the storage of data.
00036         @param force_rec (bool, optional): Option to record the forces (Fx, Fy, Mz). Defaults to True.
00037         @param def_rec (bool, optional): Option to record the deformation (theta) for ZeroLength
element. Defaults to True.
00038         @param time_rec (bool, optional): Option to record time. Defaults to True.
00039         """
00040         if self.Initialized:
00041             if not os.path.exists(data_dir):
00042                 print("Folder {} not found in this directory; creating one".format(data_dir))
00043                 os.makedirs(data_dir)
00044
00045             if time_rec:
00046                 if force_rec:
00047                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time",
"-ele", ele_ID, "force")
00048                 if def_rec:
00049                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time",
"-ele", ele_ID, "deformation")
00050             else:
00051                 if force_rec:
00052                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele",
ele_ID, "force")
00053                 if def_rec:
00054                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele",
ele_ID, "deformation")
00055             else:
00056                 print("The element is not initialized (node and/or elements not created), ID =
{}".format(ele_ID))
00057

```

## 7.33.2.2 RecordNodeDef()

```

def RecordNodeDef (
    self,
    int iNode_ID,
    int jNode_ID,
    str name_txt,
    str data_dir,
    time_rec = True )

```

Abstract method that records the deformation and time of the member's nodes associated with the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the node i.
-----------------	--------------------------

## Parameters

<i>jNode_ID</i>	(int): ID of the node j.
<i>name_txt</i>	(str): Name of the recorded data (no .txt).
<i>data_dir</i>	(str): Directory for the storage of data.
<i>time_rec</i>	(bool, optional): Option to record time. Defaults to True.

Reimplemented in [PanelZone](#), [ElasticElement](#), [SpringBasedElement](#), [ForceBasedElement](#), and [GIFBElement](#).

Definition at line 59 of file [MemberModel.py](#).

```

00059     def RecordNodeDef(self, iNode_ID: int, jNode_ID: int, name_txt: str, data_dir: str, time_rec =
      True):
00060         """
00061         Abstract method that records the deformation and time of the member's nodes associated with
the class.
00062
00063         @param iNode_ID (int): ID of the node i.
00064         @param jNode_ID (int): ID of the node j.
00065         @param name_txt (str): Name of the recorded data (no .txt).
00066         @param data_dir (str): Directory for the storage of data.
00067         @param time_rec (bool, optional): Option to record time. Defaults to True.
00068         """
00069         if self.Initialized:
00070             if not os.path.exists(data_dir):
00071                 print("Folder {} not found in this directory; creating one".format(data_dir))
00072                 os.makedirs(data_dir)
00073
00074             if time_rec:
00075                 recorder("Node", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time", "-node",
iNode_ID, jNode_ID, "-dof", 1, 2, 3, "disp")
00076             else:
00077                 recorder("Node", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-node", iNode_ID,
jNode_ID, "-dof", 1, 2, 3, "disp")
00078             else:
00079                 print("The element is not initialized (node and/or elements not created), iNode ID =
{}, jNode ID = {}".format(iNode_ID, jNode_ID))
00080
00081

```

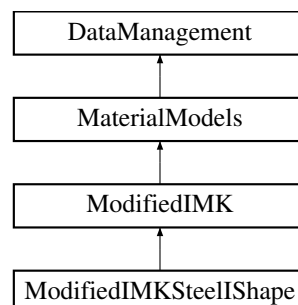
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.34 ModifiedIMK Class Reference

Class that stores functions and material properties of a steel double symmetric I-shape profile with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.

Inheritance diagram for ModifiedIMK:



## Public Member Functions

- def `__init__` (self, int `ID`, str `Type`, `d`, `bf`, `tf`, `tw`, `h_1`, `ly_mod`, `iz`, `E`, `Fy`, `Npl`, `My`, `L`, `N_G`=0, `K_factor`=3, `L_0`=-1, `L_b`=-1, `Mc`=-1, `K`=-1, `theta_u`=-1, `safety_factors`=False)  
*Constructor of the class.*
- def `Bilin` (self)  
*Generate the material model Bilin (Modified IMK) using the computed parameters.*
- def `CheckApplicability` (self)  
*Implementation of the homonym abstract method.*
- def `Computea` (self)  
*Method that computes the strain hardening ratio with the n modification.*
- def `Computea_s` (self)  
*Method that computes the modified strain hardening ratio for the spring.*
- def `ComputeK` (self)  
*Method that computes the residual strength ratio.*
- def `ComputeKe` (self)  
*Method that computes the elastic stiffness.*
- def `ComputeMc` (self)  
*Method that computes the capping moment.*
- def `ComputeMyStar` (self)  
*Method that computes the effective yield moment.*
- def `ComputeRefEnergyDissipationCap` (self)  
*Method that computes the reference energy dissipation capacity.*
- def `ComputeTheta_p` (self)  
*Method that computes the plastic rotation.*
- def `ComputeTheta_pc` (self)  
*Method that computes the post capping rotation.*
- def `ComputeTheta_u` (self)  
*Method that computes the ultimate rotation.*
- def `ComputeTheta_y` (self)  
*Method that computes the yield rotation.*
- def `Relnit` (self, `Mc`=-1, `K`=-1, `theta_u`=-1)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, `plot`=False, `block`=False)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- `a`
- `a_s`
- `bf`
- `d`
- `data`
- `E`
- `Fy`
- `gamma_rm`
- `h_1`
- `ID`
- `Initialized`

- [ly\\_mod](#)
- [iz](#)
- [K](#)
- [K\\_factor](#)
- [Ke](#)
- [L](#)
- [L\\_0](#)
- [L\\_b](#)
- [Mc](#)
- [McMy](#)
- [My](#)
- [My\\_star](#)
- [N\\_G](#)
- [Npl](#)
- [prob\\_factor](#)
- [rate\\_det](#)
- [section\\_name\\_tag](#)
- [tf](#)
- [theta\\_p](#)
- [theta\\_pc](#)
- [theta\\_u](#)
- [theta\\_y](#)
- [tw](#)
- [Type](#)

## Static Public Attributes

- float [n](#) = 10.0

### 7.34.1 Detailed Description

Class that stores functions and material properties of a steel double symmetric I-shape profile with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.

The default values are valid for a simple cantilever. For more information about the empirical model for the computation of the parameters, see Lignos Krawinkler 2011. The parameter 'n' is used as global throughout the SteelShape sections to optimise the program (given the fact that is constant everytime).

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 34 of file [MaterialModels.py](#).

### 7.34.2 Constructor & Destructor Documentation

7.34.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    str Type,
    d,
    bf,
    tf,
    tw,
    h_1,
    Iy_mod,
    iz,
    E,
    Fy,
    Npl,
    My,
    L,
    N_G = 0,
    K_factor = 3,
    L_0 = -1,
    L_b = -1,
    Mc = -1,
    K = -1,
    theta_u = -1,
    safety_factors = False )
```

Constructor of the class.

Every argument that is optional and is initialised as -1, will be computed in this class.

## Parameters

<i>ID</i>	(int): ID of the material model.
<i>Type</i>	(str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
<i>d</i>	(float): Depth of the section.
<i>bf</i>	(float): Flange's width of the section
<i>tf</i>	(float): Flange's thickness of the section
<i>tw</i>	(float): Web's thickness of the section
<i>h_1</i>	(float): Depth excluding the flange's thicknesses and the weld fillets.
<i>Iy_mod</i>	(float): n modified moment of inertia (strong axis)
<i>iz</i>	(float): Radius of gyration (weak axis).
<i>E</i>	(float): Young modulus.
<i>Fy</i>	(float): Yield strength.
<i>Npl</i>	(float): Maximal vertical axial load.
<i>My</i>	(float): Yielding moment.
<i>L</i>	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
<i>N_G</i>	(float, optional): Gravity axial load. Defaults to 0.
<i>K_factor</i>	(float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
<i>L_0</i>	(float, optional): Position of the inflection point. Defaults to -1, e.g. computed as the total length, assuming cantilever.
<i>L_b</i>	(float, optional): Maximal unbraced lateral torsional buckling length. Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing support.

## Parameters

<i>Mc</i>	(float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
<i>K</i>	(float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
<i>theta_u</i>	(float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
<i>safety_factors</i>	(bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>WrongArgument</i>	Type needs to be 'Col' or 'Beam'.
<i>NegativeValue</i>	d needs to be positive.
<i>NegativeValue</i>	bf needs to be positive.
<i>NegativeValue</i>	tf needs to be positive.
<i>NegativeValue</i>	tw needs to be positive.
<i>NegativeValue</i>	h_1 needs to be positive.
<i>NegativeValue</i>	Iy_mod needs to be positive.
<i>NegativeValue</i>	iz needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	Fy needs to be positive.
<i>NegativeValue</i>	Npl needs to be positive.
<i>NegativeValue</i>	My needs to be positive.
<i>NegativeValue</i>	L needs to be positive.
<i>NegativeValue</i>	N_G needs to be positive.
<i>NegativeValue</i>	L_0 needs to be positive if different from -1.
<i>NegativeValue</i>	L_b needs to be positive if different from -1.
<i>NegativeValue</i>	Mc needs to be positive if different from -1.
<i>NegativeValue</i>	K needs to be positive if different from -1.
<i>NegativeValue</i>	theta_u needs to be positive if different from -1.
<i>InconsistentGeometry</i>	h_1 can't be bigger than d
<i>MemberFailure</i>	N_G can't be bigger than Npl (section failure).
<i>InconsistentGeometry</i>	L_0 can't be bigger than L

Reimplemented in [ModifiedIMKSteelShape](#).

Definition at line 47 of file [MaterialModels.py](#).

```

00048         N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K = -1, theta_u = -1, safety_factors =
False):
00049         """
00050         Constructor of the class. Every argument that is optional and is initialised as -1, will be
computed in this class.
00051
00052         @param ID (int): ID of the material model.
00053         @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
00054         @param d (float): Depth of the section.
00055         @param bf (float): Flange's width of the section
00056         @param tf (float): Flange's thickness of the section
00057         @param tw (float): Web's thickness of the section
00058         @param h_1 (float): Depth excluding the flange's thicknesses and the weld fillets.
00059         @param Iy_mod (float): n modified moment of inertia (strong axis)
00060         @param iz (float): Radius of gyration (weak axis).
00061         @param E (float): Young modulus.
00062         @param Fy (float): Yield strength.
00063         @param Npl (float): Maximal vertical axial load.
00064         @param My (float): Yielding moment.
00065         @param L (float): Effective length of the element associated with this section.

```



```

00066         If the panel zone is present, exclude its dimension.
00067         @param N_G (float, optional): Gravity axial load. Defaults to 0.
00068         @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
00069         @param L_0 (float, optional): Position of the inflection point.
00070             Defaults to -1, e.g. computed as the total length, assuming cantilever.
00071         @param L_b (float, optional): Maximal unbraced lateral torsional buckling length.
00072             Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing
support.
00073         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00074         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
ComputeK.
00075         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
ComputeTheta_u.
00076         @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters
are used (not test results). Defaults to False.
00077
00078         @exception NegativeValue: ID needs to be a positive integer.
00079         @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
00080         @exception NegativeValue: d needs to be positive.
00081         @exception NegativeValue: bf needs to be positive.
00082         @exception NegativeValue: tf needs to be positive.
00083         @exception NegativeValue: tw needs to be positive.
00084         @exception NegativeValue: h_1 needs to be positive.
00085         @exception NegativeValue: Iy_mod needs to be positive.
00086         @exception NegativeValue: iz needs to be positive.
00087         @exception NegativeValue: E needs to be positive.
00088         @exception NegativeValue: Fy needs to be positive.
00089         @exception NegativeValue: Npl needs to be positive.
00090         @exception NegativeValue: My needs to be positive.
00091         @exception NegativeValue: L needs to be positive.
00092         @exception NegativeValue: N_G needs to be positive.
00093         @exception NegativeValue: L_0 needs to be positive if different from -1.
00094         @exception NegativeValue: L_b needs to be positive if different from -1.
00095         @exception NegativeValue: Mc needs to be positive if different from -1.
00096         @exception NegativeValue: K needs to be positive if different from -1.
00097         @exception NegativeValue: theta_u needs to be positive if different from -1.
00098         @exception InconsistentGeometry: h_1 can't be bigger than d
00099         @exception MemberFailure: N_G can't be bigger than Npl (section failure).
00100         @exception InconsistentGeometry: L_0 can't be bigger than L
00101         """
00102         # Check
00103         if ID < 0: raise NegativeValue()
00104         if Type != "Beam" and Type != "Col": raise WrongArgument()
00105         if d < 0: raise NegativeValue()
00106         if bf < 0: raise NegativeValue()
00107         if tf < 0: raise NegativeValue()
00108         if tw < 0: raise NegativeValue()
00109         if h_1 < 0: raise NegativeValue()
00110         if Iy_mod < 0: raise NegativeValue()
00111         if iz < 0: raise NegativeValue()
00112         if E < 0: raise NegativeValue()
00113         if Fy < 0: raise NegativeValue()
00114         if Npl < 0: raise NegativeValue()
00115         if My < 0: raise NegativeValue()
00116         if L < 0: raise NegativeValue()
00117         if N_G < 0: raise NegativeValue()
00118         if L_0 != -1 and L_0 < 0: raise NegativeValue()
00119         if L_b != -1 and L_b < 0: raise NegativeValue()
00120         if Mc != -1 and Mc < 0: raise NegativeValue()
00121         if K != -1 and K < 0: raise NegativeValue()
00122         if theta_u != -1 and theta_u < 0: raise NegativeValue()
00123         if h_1 > d: raise InconsistentGeometry()
00124         if N_G > Npl: raise MemberFailure()
00125         if L_0 > L: raise InconsistentGeometry()
00126
00127         # Arguments
00128         self.Type = Type
00129         self.ID = ID
00130         self.d = d
00131         self.bf = bf
00132         self.tf = tf
00133         self.tw = tw
00134         self.h_1 = h_1
00135         self.Iy_mod = Iy_mod
00136         self.iz = iz
00137         self.E = E
00138         self.Fy = Fy
00139         self.Npl = Npl
00140         self.My = My
00141         self.L = L
00142         self.N_G = N_G
00143         self.K_factor = K_factor
00144         self.L_0 = L if L_0 == -1 else L_0
00145         self.L_b = L if L_b == -1 else L_b
00146
00147         # Initialized the parameters that are dependent from others
00148         self.section_name_tag = "None"

```

```

00149         self.Initialized = False
00150     if safety_factors:
00151         self.gamma_rm = 1.25
00152         self.prob_factor = 1.15
00153     else:
00154         self.gamma_rm = 1.0
00155         self.prob_factor = 1.0
00156     self.ReInit(Mc, K, theta_u)
00157
00158

```

### 7.34.3 Member Function Documentation

#### 7.34.3.1 Bilin()

```

def Bilin (
    self )

```

Generate the material model Bilin (Modified IMK) using the computed parameters.

See `_Bilin` function for more information.

Definition at line 478 of file [MaterialModels.py](#).

```

00478     def Bilin(self):
00479         """
00480         Generate the material model Bilin (Modified IMK) using the computed parameters.
00481         See _Bilin function for more information.
00482         """
00483         _Bilin(self.ID, self.Ke, self.a_s, self.My_star, self.theta_p, self.theta_pc, self.K,
00484               self.theta_u, self.rate_det)
00485         self.Initialized = True
00486         self.UpdateStoredData()
00487

```

#### 7.34.3.2 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 289 of file [MaterialModels.py](#).

```

00289     def CheckApplicability(self):
00290         """
00291         Implementation of the homonym abstract method.
00292         See parent class MaterialModels for detailed information.
00293         """
00294         Check = True
00295         if self.Type == "Beam":
00296             if self.d/self.tw < 20 or self.d/self.tw > 55:
00297                 Check = False
00298                 print("The d/tw check was not fullfilled")
00299             if self.L_b/self.iz < 20 or self.L_b/self.iz > 80:
00300                 Check = False
00301                 print("The Lb/iz check was not fullfilled")
00302             if self.bf/2/self.tf < 4 or self.bf/2/self.tf > 8:

```

```

00303         Check = False
00304         print("The bf/2/tf check was not fullfilled")
00305         if self.L/self.d < 2.5 or self.L/self.d > 7:
00306             Check = False
00307             print("The check L/d was not fullfilled")
00308         if self.d < 102*mm_unit or self.d > 914*mm_unit:
00309             Check = False
00310             print("The d check was not fullfilled")
00311         if self.Fy < 240*MPa_unit or self.Fy > 450*MPa_unit:
00312             Check = False
00313             print("The Fy check was not fullfilled")
00314         else:
00315             if self.h_l/self.tw < 3.71 or self.d/self.tw > 57.5:
00316                 Check = False
00317                 print("The h_l/tw check was not fullfilled")
00318             if self.L_b/self.iz < 38.4 or self.L_b/self.iz > 120:
00319                 Check = False
00320                 print("The L_b/iz check was not fullfilled")
00321             if self.N_G/self.Npl < 0 or self.N_G/self.Npl > 0.75:
00322                 Check = False
00323                 print("The NG/Npl check was not fullfilled")
00324         if not Check:
00325             print("The validity of the equations is not fullfilled.")
00326             print("!!!!!! WARNING !!!!!!! Check material model of Modified IMK, ID=", self.ID)
00327             print("")
00328

```

### 7.34.3.3 Computea()

```

def Computea (
    self )

```

Method that computes the strain hardening ratio with the n modification.

#### Returns

float: Strain hardening ratio.

Definition at line 337 of file [MaterialModels.py](#).

```

00337     def Computea(self):
00338         """
00339         Method that computes the strain hardening ratio with the n modification.
00340
00341         @returns float: Strain hardening ratio.
00342         """
00343         # strain hardening ratio of spring
00344         return (n+1.0)*self.My_star*(self.McMy-1.0)/(self.Ke*self.theta_p)
00345

```

### 7.34.3.4 Computea\_s()

```

def Computea_s (
    self )

```

Method that computes the modified strain hardening ratio for the spring.

For more info see Ibarra & Krawinkler 2005.

#### Returns

float: Strain hardening ratio.

Definition at line 346 of file [MaterialModels.py](#).

```

00346     def Computea_s(self):
00347         """
00348         Method that computes the modified strain hardening ratio for the spring.
00349         For more info see Ibarra & Krawinkler 2005.
00350
00351         @returns float: Strain hardening ratio.
00352         """
00353         return self.a/(1.0+n*(1.0-self.a))
00354

```

### 7.34.3.5 ComputeK()

```
def ComputeK (
    self )
```

Method that computes the residual strength ratio.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

#### Returns

float: Residual strength ratio.

Definition at line 384 of file [MaterialModels.py](#).

```
00384     def ComputeK(self):
00385         """
00386         Method that computes the residual strength ratio.
00387         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00388
00389         @returns float: Residual strength ratio.
00390         """
00391         if self.Type == "Beam":
00392             return 0.4
00393         else:
00394             tmp = 0.5-0.4*self.N_G/self.Npl
00395             return max(tmp, 0)
00396
```

### 7.34.3.6 ComputeKe()

```
def ComputeKe (
    self )
```

Method that computes the elastic stiffness.

#### Returns

float: The stiffness

Definition at line 329 of file [MaterialModels.py](#).

```
00329     def ComputeKe(self):
00330         """
00331         Method that computes the elastic stiffness.
00332
00333         @returns float: The stiffness
00334         """
00335         return self.K_factor*n*self.E*self.Iy_mod/self.L
00336
```

### 7.34.3.7 ComputeMc()

```
def ComputeMc (
    self )
```

Method that computes the capping moment.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

#### Returns

float: Capping moment.

Definition at line 370 of file [MaterialModels.py](#).

```
00370     def ComputeMc(self):
00371         """
00372         Method that computes the capping moment.
00373         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00374
00375         @returns float: Capping moment.
00376         """
00377         if self.Type == "Beam":
00378             return self.My_star*1.11
00379             # For RBS: My_star*1.09
00380         else:
00381             tmp =
00382             12.5*(self.h_l/self.tw)**(-0.2)*(self.L_b/self.iz)**(-0.4)*(1-self.N_G/self.Npl)**0.4
00383             return max(min(1.3, tmp), 1.0)*self.My_star
```

### 7.34.3.8 ComputeMyStar()

```
def ComputeMyStar (
    self )
```

Method that computes the effective yield moment.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

#### Returns

float: Effective yield moment.

Definition at line 355 of file [MaterialModels.py](#).

```
00355     def ComputeMyStar(self):
00356         """
00357         Method that computes the effective yield moment.
00358         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00359
00360         @returns float: Effective yield moment.
00361         """
00362         if self.Type == "Beam":
00363             return self.prob_factor*self.My*self.gamma_rm*1.1
00364         else:
00365             if self.N_G/self.Npl > 0.2:
00366                 return 1.15*self.prob_factor*self.My*self.gamma_rm*(1-self.N_G/self.Npl)*9.0/8.0
00367             else:
00368                 return 1.15*self.prob_factor*self.My*self.gamma_rm*(1-self.N_G/2.0/self.Npl)
00369
```

### 7.34.3.9 ComputeRefEnergyDissipationCap()

```
def ComputeRefEnergyDissipationCap (
    self )
```

Method that computes the reference energy dissipation capacity.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

#### Returns

float: Reference energy dissipation capacity.

Definition at line 456 of file [MaterialModels.py](#).

```
00456     def ComputeRefEnergyDissipationCap(self):
00457         """
00458         Method that computes the reference energy dissipation capacity.
00459         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00460
00461         @returns float: Reference energy dissipation capacity.
00462         """
00463         if self.Type == "Beam":
00464             if self.d < 533.0*mm_unit:
00465                 return
00466             495.0*(self.h_l/self.tw)**(-1.34)*(self.bf/2.0/self.tf)**(-0.595)*(self.Fy/(355.0*MPa_unit))**(-0.360)
00467             else:
00468                 return
00469             536.0*(self.h_l/self.tw)**(-1.26)*(self.bf/2.0/self.tf)**(-0.525)*(self.L_b/self.iz)**(-0.130)*(self.Fy/(355.0*MPa_unit))
00470             # With RBS: ...
00471             else:
00472                 if self.N_G/self.Npl > 0.35:
00473                     tmp =
00474                     268000.0*(self.h_l/self.tw)**(-2.30)*(self.L_b/self.iz)**(-1.130)*(1.0-self.N_G/self.Npl)**(1.19)
00475                     return min(tmp, 3.0)
00476                 else:
00477                     tmp =
00478                     25000.0*(self.h_l/self.tw)**(-2.14)*(self.L_b/self.iz)**(-0.53)*(1.0-self.N_G/self.Npl)**(4.92)
00479                     return min(tmp, 3.0)
```

### 7.34.3.10 ComputeTheta\_p()

```
def ComputeTheta_p (
    self )
```

Method that computes the plastic rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

## Returns

float: Plastic rotation.

Definition at line 406 of file [MaterialModels.py](#).

```
00406     def ComputeTheta_p(self):
00407         """
00408         Method that computes the plastic rotation.
00409         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00410
00411         @returns float: Plastic rotation.
00412         """
00413         if self.Type == "Beam":
00414             if self.d < 533.0*mm_unit:
00415                 return
00416             0.0865*(self.h_1/self.tw)**(-0.365)*(self.bf/2.0/self.tf)**(-0.14)*(self.L_0/self.d)**(0.34)*(self.d/(533.0*mm_unit))*
00417             else:
00418                 return
00419             0.318*(self.h_1/self.tw)**(-0.550)*(self.bf/2.0/self.tf)**(-0.345)*(self.L_0/self.d)**(0.090)*(self.L_b/self.iz)**(-0.
00420             # With RBS: ...
00421         else:
00422             tmp =
00423             294.0*(self.h_1/self.tw)**(-1.7)*(self.L_b/self.iz)**(-0.7)*(1.0-self.N_G/self.Npl)**(1.6) #
00424             *(self.E/self.Fy/gamma_rm)**(0.2) # EC8
00425             if tmp > 0.2:
00426                 tmp = 0.2
00427             # if tmp > self.theta_u-self.theta_y:
00428             #     tmp = (self.theta_u-self.theta_y)*0.799 # convergence issue
00429             return tmp
```

### 7.34.3.11 ComputeTheta\_pc()

```
def ComputeTheta_pc (
    self )
```

Method that computes the post capping rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

## Returns

float: Post capping rotation.

Definition at line 427 of file [MaterialModels.py](#).

```
00427     def ComputeTheta_pc(self):
00428         """
00429         Method that computes the post capping rotation.
00430         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00431
00432         @returns float: Post capping rotation.
00433         """
00434         if self.Type == "Beam":
00435             if self.d < 533.0*mm_unit:
00436                 return
00437             5.63*(self.h_1/self.tw)**(-0.565)*(self.bf/2.0/self.tf)**(-0.800)*(self.d/(533.0*mm_unit))**(-0.280)*(self.Fy/(355.0*M
00438             else:
00439                 return
00440             7.50*(self.h_1/self.tw)**(-0.610)*(self.bf/2.0/self.tf)**(-0.710)*(self.L_b/self.iz)**(-0.110)*(self.d/(533.0*mm_unit))
00441             # With RBS: ...
00442         else:
00443             tmp =
00444             90.0*(self.h_1/self.tw)**(-0.8)*(self.L_b/self.iz)**(-0.8)*(1.0-self.N_G/self.Npl)**(2.5) #
00445             *(self.E/self.Fy/gamma_rm)**(0.07) # EC8
00446             return min(tmp, 0.3)
```

### 7.34.3.12 ComputeTheta\_u()

```
def ComputeTheta_u (
    self )
```

Method that computes the ultimate rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

#### Returns

float: Ultimate rotation.

Definition at line 444 of file [MaterialModels.py](#).

```
00444     def ComputeTheta_u(self):
00445         """
00446         Method that computes the ultimate rotation.
00447         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00448
00449         @returns float: Ultimate rotation.
00450         """
00451         if self.Type == "Beam":
00452             return 0.2
00453         else:
00454             return 0.15
00455
```

### 7.34.3.13 ComputeTheta\_y()

```
def ComputeTheta_y (
    self )
```

Method that computes the yield rotation.

For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.

#### Returns

float: Yield rotation.

Definition at line 397 of file [MaterialModels.py](#).

```
00397     def ComputeTheta_y(self):
00398         """
00399         Method that computes the yield rotation.
00400         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00401
00402         @returns float: Yield rotation.
00403         """
00404         return self.My_star/self.Ke*(n+1)
00405
```

### 7.34.3.14 ReInit()

```
def ReInit (
    self,
    Mc = -1,
    K = -1,
    theta_u = -1 )
```

Implementation of the homonym abstract method.

See parent class DataManagement for detailed information.



## Parameters

<i>Mc</i>	(float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
<i>K</i>	(float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
<i>theta_u</i>	(float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.

Definition at line 160 of file [MaterialModels.py](#).

```

00160     def ReInit(self, Mc = -1, K = -1, theta_u = -1):
00161         """
00162         Implementation of the homonym abstract method.
00163         See parent class DataManagement for detailed information.
00164
00165         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00166         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
00167         ComputeK.
00168         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
00169         ComputeTheta_u.
00170         """
00171         # Precompute some members
00172         self.My_star = self.ComputeMyStar()
00173
00174         # Arguments
00175         self.Mc = self.ComputeMc() if Mc == -1 else Mc
00176         self.K = self.ComputeK() if K == -1 else K
00177         self.theta_u = self.ComputeTheta_u() if theta_u == -1 else theta_u
00178
00179         # Check applicability
00180         self.CheckApplicability()
00181
00182         # Members
00183         self.Ke = self.ComputeKe()
00184         self.theta_y = self.ComputeTheta_y()
00185         self.theta_p = self.ComputeTheta_p()
00186         self.theta_pc = self.ComputeTheta_pc()
00187         self.McMy = self.Mc/self.My_star
00188         self.rate_det = self.ComputeRefEnergyDissipationCap()
00189         self.a = self.Computea()
00190         self.a_s = self.Computea_s()
00191         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
00192
00193         # Data storage for loading/saving
00194         self.UpdateStoredData()

```

## 7.34.3.15 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

## Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 237 of file [MaterialModels.py](#).

```

00237     def ShowInfo(self, plot = False, block = False):
00238         """
00239         Implementation of the homonym abstract method.
00240         See parent class DataManagement for detailed information.
00241
00242         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
00243         False.
00244         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00245         """
00246         Mr = self.K*self.My_star
00247         theta_p_plot = self.theta_p
00248         if self.theta_p > self.theta_u-self.theta_y:
00249             theta_p_plot = self.theta_u-self.theta_y
00250         theta_r = self.theta_y + theta_p_plot + self.theta_pc*(1.0-Mr/self.Mc)
00251         if theta_r > self.theta_u:
00252             theta_r = self.theta_u
00253             Mr = self.Mc*(1.0-1.0/self.theta_pc*(self.theta_u-self.theta_y-theta_p_plot))
00254
00255         print("")
00256         print("Requested info for Modified IMK (Ibarra-Medina-Krawinkler) material model Parameters,
ID = {}".format(self.ID))
00257         print("Section associated: {}".format(self.section_name_tag))
00258         print('theta y = {}'.format(self.theta_y))
00259         print('theta p = {}'.format(self.theta_p))
00260         print('theta r = {}'.format(theta_r))
00261         print('theta pc = {}'.format(self.theta_pc))
00262         print('theta u = {}'.format(self.theta_u))
00263         print('My star = {} kNm'.format(self.My_star/kNm_unit))
00264         print('Mc = {} kNm'.format(self.Mc/kNm_unit))
00265         print('Mr = {} kNm'.format(Mr/kNm_unit))
00266         print('a = {} '.format(self.a))
00267         print('as = {} '.format(self.a_s))
00268         print('lambda (deterioration rate) = {} '.format(self.rate_det))
00269         print("")
00270
00271         if plot:
00272             # Data for plotting
00273             x_axis = np.array([0.0, self.theta_y, self.theta_y + theta_p_plot, theta_r, self.theta_u,
self.theta_u])
00274             x_axis2 = np.array([self.theta_y + theta_p_plot, self.theta_y + theta_p_plot +
self.theta_pc])
00275             y_axis = np.array([0.0, self.My_star, self.Mc, Mr, Mr, 0.0])/kNm_unit
00276             y_axis2 = np.array([self.Mc, 0.0])/kNm_unit
00277
00278             fig, ax = plt.subplots()
00279             ax.plot(x_axis, y_axis, 'k-')
00280             ax.plot(x_axis2, y_axis2, 'k--')
00281
00282             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
00283                   title='Modified IMK deterioration model (ID={})'.format(self.ID))
00284             ax.grid()
00285
00286             if block:
00287                 plt.show()
00288

```

### 7.34.3.16 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 195 of file [MaterialModels.py](#).

```

00195     def UpdateStoredData(self):
00196         """
00197         Implementation of the homonym abstract method.
00198         See parent class DataManagement for detailed information.
00199         """
00200         self.data = [{"INFO_TYPE", "ModifiedIMK"}, # Tag for differentiating different data
00201                     ["ID", self.ID],

```

```

00202         ["section_name_tag", self.section_name_tag],
00203         ["Type", self.Type],
00204         ["d", self.d],
00205         ["bf", self.bf],
00206         ["tf", self.tf],
00207         ["tw", self.tw],
00208         ["h_l", self.h_l],
00209         ["Iy_mod", self.Iy_mod],
00210         ["iz", self.iz],
00211         ["E", self.E],
00212         ["Fy", self.Fy],
00213         ["L", self.L],
00214         ["N_G", self.N_G],
00215         ["K_factor", self.K_factor],
00216         ["Ke", self.Ke],
00217         ["L_0", self.L_0],
00218         ["L_b", self.L_b],
00219         ["gamma_rm", self.gamma_rm],
00220         ["prob_factor", self.prob_factor],
00221         ["Npl", self.Npl],
00222         ["My", self.My],
00223         ["My_star", self.My_star],
00224         ["Mc", self.Mc],
00225         ["McMy", self.McMy],
00226         ["K", self.K],
00227         ["theta_y", self.theta_y],
00228         ["theta_p", self.theta_p],
00229         ["theta_pc", self.theta_pc],
00230         ["theta_u", self.theta_u],
00231         ["rate_det", self.rate_det],
00232         ["a", self.a],
00233         ["a_s", self.a_s],
00234         ["Initialized", self.Initialized]]
00235
00236

```

### 7.34.4 Member Data Documentation

#### 7.34.4.1 a

a

Definition at line 187 of file [MaterialModels.py](#).

#### 7.34.4.2 a\_s

a\_s

Definition at line 188 of file [MaterialModels.py](#).

#### 7.34.4.3 bf

bf

Definition at line 131 of file [MaterialModels.py](#).

**7.34.4.4 d**

d

Definition at line 130 of file [MaterialModels.py](#).

**7.34.4.5 data**

data

Definition at line 200 of file [MaterialModels.py](#).

**7.34.4.6 E**

E

Definition at line 137 of file [MaterialModels.py](#).

**7.34.4.7 Fy**

Fy

Definition at line 138 of file [MaterialModels.py](#).

**7.34.4.8 gamma\_rm**

gamma\_rm

Definition at line 151 of file [MaterialModels.py](#).

**7.34.4.9 h\_1**

h\_1

Definition at line 134 of file [MaterialModels.py](#).

#### 7.34.4.10 ID

ID

Definition at line 129 of file [MaterialModels.py](#).

#### 7.34.4.11 Initialized

Initialized

Definition at line 149 of file [MaterialModels.py](#).

#### 7.34.4.12 Iy\_mod

Iy\_mod

Definition at line 135 of file [MaterialModels.py](#).

#### 7.34.4.13 iz

iz

Definition at line 136 of file [MaterialModels.py](#).

#### 7.34.4.14 K

K

Definition at line 174 of file [MaterialModels.py](#).

#### 7.34.4.15 K\_factor

K\_factor

Definition at line 143 of file [MaterialModels.py](#).

**7.34.4.16 Ke**

Ke

Definition at line 181 of file [MaterialModels.py](#).

**7.34.4.17 L**

L

Definition at line 141 of file [MaterialModels.py](#).

**7.34.4.18 L\_0**

L\_0

Definition at line 144 of file [MaterialModels.py](#).

**7.34.4.19 L\_b**

L\_b

Definition at line 145 of file [MaterialModels.py](#).

**7.34.4.20 Mc**

Mc

Definition at line 173 of file [MaterialModels.py](#).

**7.34.4.21 McMy**

McMy

Definition at line 185 of file [MaterialModels.py](#).

#### 7.34.4.22 My

My

Definition at line 140 of file [MaterialModels.py](#).

#### 7.34.4.23 My\_star

My\_star

Definition at line 170 of file [MaterialModels.py](#).

#### 7.34.4.24 n

```
float n = 10.0 [static]
```

Definition at line 45 of file [MaterialModels.py](#).

#### 7.34.4.25 N\_G

N\_G

Definition at line 142 of file [MaterialModels.py](#).

#### 7.34.4.26 Npl

Npl

Definition at line 139 of file [MaterialModels.py](#).

#### 7.34.4.27 prob\_factor

prob\_factor

Definition at line 152 of file [MaterialModels.py](#).

#### 7.34.4.28 rate\_det

rate\_det

Definition at line 186 of file [MaterialModels.py](#).

#### 7.34.4.29 section\_name\_tag

section\_name\_tag

Definition at line 148 of file [MaterialModels.py](#).

#### 7.34.4.30 tf

tf

Definition at line 132 of file [MaterialModels.py](#).

#### 7.34.4.31 theta\_p

theta\_p

Definition at line 183 of file [MaterialModels.py](#).

#### 7.34.4.32 theta\_pc

theta\_pc

Definition at line 184 of file [MaterialModels.py](#).

#### 7.34.4.33 theta\_u

theta\_u

Definition at line 175 of file [MaterialModels.py](#).



**7.34.4.34 theta\_y**`theta_y`

Definition at line 182 of file [MaterialModels.py](#).

**7.34.4.35 tw**`tw`

Definition at line 133 of file [MaterialModels.py](#).

**7.34.4.36 Type**`Type`

Definition at line 128 of file [MaterialModels.py](#).

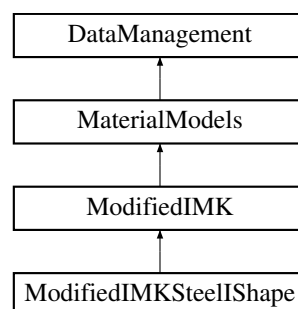
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

**7.35 ModifiedIMKSteelShape Class Reference**

Class that is the children of [ModifiedIMK](#) and combine the class `SteelShape` (section) to retrieve the information needed.

Inheritance diagram for `ModifiedIMKSteelShape`:

**Public Member Functions**

- `def __init__(self, ID, SteelShape section, N_G=0, K_factor=3, L_0=-1, L_b=-1, Mc=-1, K=-1, theta_u=-1, safety_factors=False)`  
*Constructor of the class.*

## Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

## Additional Inherited Members

### 7.35.1 Detailed Description

Class that is the children of [ModifiedIMK](#) and combine the class `SteelIShape` (`section`) to retrieve the information needed.

#### Parameters

<a href="#">ModifiedIMK</a>	Parent class.
-----------------------------	---------------

Definition at line 488 of file [MaterialModels.py](#).

### 7.35.2 Constructor & Destructor Documentation

#### 7.35.2.1 `__init__()`

```
def __init__ (
    self,
    ID,
    SteelIShape section,
    N_G = 0,
    K_factor = 3,
    L_0 = -1,
    L_b = -1,
    Mc = -1,
    K = -1,
    theta_u = -1,
    safety_factors = False )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class `SteelIShape`. Every argument that is optional and is initialised as -1, will be computed in this class. The copy of the section passed is stored in the member variable `self.section`.

#### Parameters

<i>ID</i>	(int): ID of the material model.
<i>section</i>	( <code>SteelIShape</code> ): Object that store informations for a steel I shpae section.

## Parameters

<i>N_G</i>	(float, optional): Gravity axial load. Defaults to 0.
<i>K_factor</i>	(float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
<i>L_0</i>	(float, optional): Position of the inflection point. Defaults to -1, e.g. computed as the total length, assuming cantilever.
<i>L_b</i>	(float, optional):Maximal unbraced lateral torsional buckling length. Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing support.
<i>Mc</i>	(float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
<i>K</i>	(float, optional): Residual strength ratio. Defaults to -1, e.g. computed in ComputeK.
<i>theta_u</i>	(float, optional): Ultimate rotation. Defaults to -1, e.g. computed in ComputeTheta_u.
<i>safety_factors</i>	(bool, optional): Safety factors used if standard mechanical parameters are used (not test results). Defaults to False.

Reimplemented from [ModifiedIMK](#).

Definition at line 494 of file [MaterialModels.py](#).

```

00494     def __init__(self, ID, section: SteelIShape, N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K
      = -1, theta_u = -1, safety_factors = False):
00495         """
00496         Constructor of the class. It passes the arguments into the parent class to generate the
      combination of the parent class
00497         and the section class SteelIShape.
00498         Every argument that is optional and is initialised as -1, will be computed in this class.
00499         The copy of the section passed is stored in the member variable self.section.
00500
00501         @param ID (int): ID of the material model.
00502         @param section (SteelIShape): Object that store informations for a steel I shpae section.
00503         @param N_G (float, optional): Gravity axial load. Defaults to 0.
00504         @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
00505         @param L_0 (float, optional): Position of the inflection point.
00506             Defaults to -1, e.g. computed as the total length, assuming cantilever.
00507         @param L_b (float, optional):Maximal unbraced lateral torsional buckling length.
00508             Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing
      support.
00509         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00510         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
      ComputeK.
00511         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
      ComputeTheta_u.
00512         @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters
      are used (not test results). Defaults to False.
00513         """
00514         self.section = deepcopy(section)
00515         super().__init__(ID, section.Type, section.d, section.bf, section.tf, section.tw, section.h_1,
00516             section.Iy_mod, section.iz, section.E, section.Fy, section.Npl, section.My, section.L,
      N_G,
00517             K_factor, L_0, L_b, Mc, K, theta_u, safety_factors)
00518         self.section_name_tag = section.name_tag
00519         self.UpdateStoredData()
00520
00521

```

## 7.35.3 Member Data Documentation

### 7.35.3.1 section

section

Definition at line 514 of file [MaterialModels.py](#).

### 7.35.3.2 section\_name\_tag

section\_name\_tag

Definition at line 518 of file [MaterialModels.py](#).

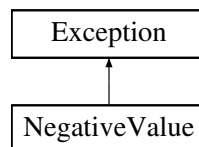
The documentation for this class was generated from the following file:

- [/media/carminе/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.36 NegativeValue Class Reference

Exception class for the "negative value (argument or result)" error.

Inheritance diagram for NegativeValue:



### 7.36.1 Detailed Description

Exception class for the "negative value (argument or result)" error.

Definition at line 16 of file [ErrorHandling.py](#).

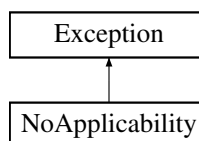
The documentation for this class was generated from the following file:

- [/media/carminе/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.37 NoApplicability Class Reference

Exception class for the "no applicability of formula of theory" error.

Inheritance diagram for NoApplicability:



### 7.37.1 Detailed Description

Exception class for the "no applicability of formula of theory" error.

Definition at line 47 of file [ErrorHandling.py](#).

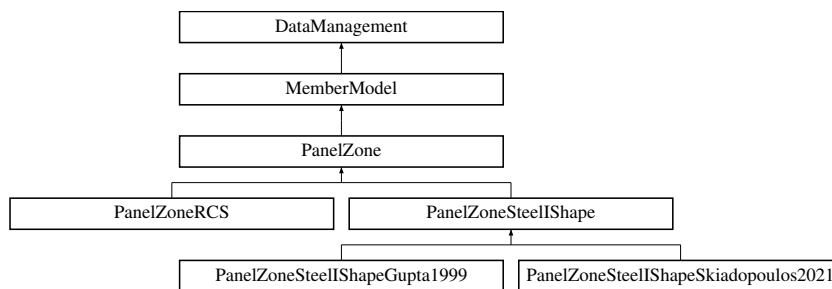
The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[ErrorHandling.py](#)

## 7.38 PanelZone Class Reference

Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for PanelZone:



### Public Member Functions

- def `__init__` (self, int [master\\_node\\_ID](#), int [mid\\_panel\\_zone\\_width](#), int [mid\\_panel\\_zone\\_height](#), E, A\_rigid, I\_rigid, int [geo\\_transf\\_ID](#), int [mat\\_ID](#), [pin\\_corners](#)=True)  
*Constructor of the class.*
- def [CreateMember](#) (self)  
*Method that initialises the member by calling the OpenSeesPy commands through various functions.*
- def [Record](#) (self, str name\_txt, str data\_dir, force\_rec=True, def\_rec=True, time\_rec=True)  
*Implementation of the homonym abstract method.*
- def [RecordNodeDef](#) (self, str name\_txt, str data\_dir, time\_rec=True)  
*Implementation of the homonym abstract method.*
- def [Relnit](#) (self)  
*Implementation of the homonym abstract method.*
- def [ShowInfo](#) (self, plot=False, block=False)  
*Implementation of the homonym abstract method.*
- def [UpdateStoredData](#) (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- [A\\_rigid](#)
- [beam\\_section\\_name\\_tag](#)
- [col\\_section\\_name\\_tag](#)
- [data](#)
- [E](#)
- [element\\_array](#)
- [geo\\_transf\\_ID](#)
- [I\\_rigid](#)
- [Initialized](#)
- [iNode\\_ID](#)
- [jNode\\_ID](#)
- [master\\_node\\_ID](#)
- [mat\\_ID](#)
- [mid\\_panel\\_zone\\_height](#)
- [mid\\_panel\\_zone\\_width](#)
- [pin\\_corners](#)
- [spring\\_ID](#)

### 7.38.1 Detailed Description

Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

#### Parameters

<a href="#">MemberModel</a>	Parent abstract class.
-----------------------------	------------------------

Definition at line 94 of file [MemberModel.py](#).

### 7.38.2 Constructor & Destructor Documentation

#### 7.38.2.1 `__init__()`

```
def __init__ (
    self,
    int master_node_ID,
    mid_panel_zone_width,
    mid_panel_zone_height,
    E,
    A_rigid,
    I_rigid,
    int geo_transf_ID,
    int mat_ID,
    pin_corners = True )
```

Constructor of the class.

## Parameters

<i>master_node_ID</i>	(int): ID of the master node (central top node that should be a grid node).
<i>mid_panel_zone_width</i>	(float): Mid panel zone width.
<i>mid_panel_zone_height</i>	(float): Mid panel zone height.
<i>E</i>	(float): Young modulus.
<i>A_rigid</i>	(float): A very rigid area.
<i>I_rigid</i>	(float): A very rigid moment of inertia.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>mat_ID</i>	(int): ID of the material model for the panel zone spring.
<i>pin_corners</i>	(bool, optional): Option to pin the corners (xy03/xy04, xy06/xy07, xy09/xy10) or not. Used for RCS models. Defaults to True.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	mid_panel_zone_width needs to be positive.
<i>NegativeValue</i>	mid_panel_zone_height needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	A_rigid needs to be positive.
<i>NegativeValue</i>	I_rigid needs to be positive.
<i>NegativeValue</i>	geo_tranf_ID needs to be a positive integer.
<i>NegativeValue</i>	mat_ID needs to be a positive integer.

Reimplemented in [PanelZoneRCS](#), [PanelZoneSteelIShape](#), [PanelZoneSteelIShapeGupta1999](#), and [PanelZoneSteelIShapeSkiadopoulos](#)

Definition at line 100 of file [MemberModel.py](#).

```

00100     def __init__(self, master_node_ID: int, mid_panel_zone_width, mid_panel_zone_height, E, A_rigid,
00101                  I_rigid, geo_transf_ID: int, mat_ID: int, pin_corners = True):
00102         """
00103             Constructor of the class.
00104
00105             @param master_node_ID (int): ID of the master node (central top node that should be a grid
00106             node).
00107             @param mid_panel_zone_width (float): Mid panel zone width.
00108             @param mid_panel_zone_height (float): Mid panel zone height.
00109             @param E (float): Young modulus.
00110             @param A_rigid (float): A very rigid area.
00111             @param I_rigid (float): A very rigid moment of inertia.
00112             @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00113             documentation).
00114             @param mat_ID (int): ID of the material model for the panel zone spring.
00115             @param pin_corners (bool, optional): Option to pin the corners (xy03/xy04, xy06/xy07,
00116             xy09/xy10) or not. Used for RCS models. Defaults to True.
00117
00118             @exception NegativeValue: ID needs to be a positive integer.
00119             @exception NegativeValue: mid_panel_zone_width needs to be positive.
00120             @exception NegativeValue: mid_panel_zone_height needs to be positive.
00121             @exception NegativeValue: E needs to be positive.
00122             @exception NegativeValue: A_rigid needs to be positive.
00123             @exception NegativeValue: I_rigid needs to be positive.
00124             @exception NegativeValue: geo_tranf_ID needs to be a positive integer.
00125             @exception NegativeValue: mat_ID needs to be a positive integer.
00126         """
00127         # Check
00128         if master_node_ID < 1: raise NegativeValue()
00129         # if master_node_ID > 99: raise WrongNodeIDConvention(master_node_ID)
00130         if mid_panel_zone_width < 0: raise NegativeValue()
00131         if mid_panel_zone_height < 0: raise NegativeValue()
00132         if E < 0: raise NegativeValue()
00133         if A_rigid < 0: raise NegativeValue()
00134         if I_rigid < 0: raise NegativeValue()
00135         if geo_transf_ID > 1: raise NegativeValue()
00136         if mat_ID < 0: raise NegativeValue()

```

```

00133
00134     # Arguments
00135     self.master_node_ID = master_node_ID
00136     self.mid_panel_zone_width = mid_panel_zone_width
00137     self.mid_panel_zone_height = mid_panel_zone_height
00138     self.E = E
00139     self.A_rigid = A_rigid
00140     self.I_rigid = I_rigid
00141     self.geo_transf_ID = geo_transf_ID
00142     self.mat_ID = mat_ID
00143     self.pin_corners = pin_corners
00144
00145     # Initialized the parameters that are dependent from others
00146     self.col_section_name_tag = "None"
00147     self.beam_section_name_tag = "None"
00148     self.Initialized = False
00149     self.ReInit()
00150
00151

```

## 7.38.3 Member Function Documentation

### 7.38.3.1 CreateMember()

```

def CreateMember (
    self )

```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 220 of file [MemberModel.py](#).

```

00220     def CreateMember(self):
00221         """
00222         Method that initialises the member by calling the OpenSeesPy commands through various
00223         functions.
00224         """
00225         # Define nodes
00226         DefinePanelZoneNodes(self.master_node_ID, self.mid_panel_zone_width,
00227                             self.mid_panel_zone_height)
00228         xy1 = IDConvention(self.master_node_ID, 1)
00229         xy01 = IDConvention(self.master_node_ID, 1, 1)
00230         xy03 = IDConvention(self.master_node_ID, 3, 1)
00231         xy04 = IDConvention(self.master_node_ID, 4, 1)
00232         xy06 = IDConvention(self.master_node_ID, 6, 1)
00233         xy07 = IDConvention(self.master_node_ID, 7, 1)
00234         xy09 = IDConvention(self.master_node_ID, 9, 1)
00235         xy10 = IDConvention(self.master_node_ID, 10)
00236
00237         # Define rigid elements
00238         self.element_array = DefinePanelZoneElements(self.master_node_ID, self.E, self.A_rigid,
00239                                                     self.I_rigid, self.geo_transf_ID)
00240
00241         # Define zero length element
00242         self.spring_ID = IDConvention(xy1, xy01)
00243         RotationalSpring(self.spring_ID, xy1, xy01, self.mat_ID)
00244         self.element_array.append([self.spring_ID, xy1, xy01])
00245         self.iNode_ID = xy1
00246         self.jNode_ID = xy01
00247
00248         # Pin connections
00249         if self.pin_corners:
00250             Pin(xy03, xy04)
00251             Pin(xy06, xy07)
00252             Pin(xy09, xy10)
00253
00254         # Update class
00255         self.Initialized = True
00256         self.UpdateStoredData()

```



### 7.38.3.2 Record()

```
def Record (
    self,
    str name_txt,
    str data_dir,
    force_rec = True,
    def_rec = True,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 256 of file [MemberModel.py](#).

```
00256     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00257         """
00258         Implementation of the homonym abstract method.
00259         See parent class MemberModel for detailed information.
00260         """
00261         super().Record(self.spring_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
00262                        time_rec=time_rec)
00262
00263
```

### 7.38.3.3 RecordNodeDef()

```
def RecordNodeDef (
    self,
    str name_txt,
    str data_dir,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 264 of file [MemberModel.py](#).

```
00264     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00265         """
00266         Implementation of the homonym abstract method.
00267         See parent class MemberModel for detailed information.
00268         """
00269         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
00270
00271
```

### 7.38.3.4 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 152 of file [MemberModel.py](#).

```
00152     def ReInit(self):
00153         """
00154         Implementation of the homonym abstract method.
00155         See parent class DataManagement for detailed information.
00156         """
00157         # Arguments
00158         self.spring_ID = -1
00159
00160         # Members
00161         if self.col_section_name_tag != "None": self.col_section_name_tag = self.col_section_name_tag
+ " (modified)"
00162         if self.beam_section_name_tag != "None": self.beam_section_name_tag =
self.beam_section_name_tag + " (modified)"
00163
00164         # Data storage for loading/saving
00165         self.UpdateStoredData()
00166
00167
```

### 7.38.3.5 ShowInfo()

```
def ShowInfo (
    self,
    plot = False,
    block = False )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 189 of file [MemberModel.py](#).

```
00189     def ShowInfo(self, plot = False, block = False):
00190         """
00191         Implementation of the homonym abstract method.
00192         See parent class DataManagement for detailed information.
00193
00194         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00195         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00196         """
00197         print("")
00198         print("Requested info for Panel Zone member model, master node ID =
{}".format(self.master_node_ID))
00199         print("Section associated, column: {}".format(self.col_section_name_tag))
00200         print("Section associated, beam: {}".format(self.beam_section_name_tag))
00201         print("Material model of the panel zone ID = {}".format(self.mat_ID))
```

```

00202     print("Spring ID = {} (if -1, not defined yet)".format(self.spring_ID))
00203     print("Mid panel zone width = {} mm".format(self.mid_panel_zone_width/mm_unit))
00204     print("Mid panel zone height = {} mm".format(self.mid_panel_zone_height/mm_unit))
00205     print("Young modulus E = {} GPa".format(self.E/GPa_unit))
00206     print("Area of the elements (rigid) = {} mm2".format(self.A_rigid/mm2_unit))
00207     print("Moment of inertia of the elements (strong axis, rigid) = {}
mm4".format(self.I_rigid/mm4_unit))
00208     print("Geometric transformation = {}".format(self.geo_transf_ID))
00209     print("")
00210
00211     if plot:
00212         if self.Initialized:
00213             plot_member(self.element_array, "Panel zone, ID = {}".format(self.master_node_ID))
00214             if block:
00215                 plt.show()
00216         else:
00217             print("The panel zone is not initialized (node and elements not created) for master
node ID = {}".format(self.master_node_ID))
00218
00219

```

### 7.38.3.6 UpdateStoredData()

```
def UpdateStoredData (
    self )
```

### Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 169 of file MemberModel.py.

```
00169 def UpdateStoredData(self):
00170     """
00171     Implementation of the homonym abstract method.
00172     See parent class DataManagement for detailed information.
00173     """
00174     self.data = [{"INFO_TYPE": "PanelZone", # Tag for differentiating different data
00175                  ["master_node_ID", self.master_node_ID],
00176                  ["col_section_name_tag", self.col_section_name_tag],
00177                  ["beam_section_name_tag", self.beam_section_name_tag],
00178                  ["mat_ID", self.mat_ID],
00179                  ["spring_ID", self.spring_ID],
00180                  ["mid_panel_zone_width", self.mid_panel_zone_width],
00181                  ["mid_panel_zone_height", self.mid_panel_zone_height],
00182                  ["E", self.E],
00183                  ["A_rigid", self.A_rigid],
00184                  ["I_rigid", self.I_rigid],
00185                  ["tranf_ID", self.geo_transf_ID],
00186                  ["Initialized", self.Initialized]]
00187
00188
```

#### 7.38.4 Member Data Documentation

#### 7.38.4.1 A rigid

A\_rigid

Definition at line 139 of file MemberModel.py.

#### 7.38.4.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 147 of file [MemberModel.py](#).

#### 7.38.4.3 col\_section\_name\_tag

col\_section\_name\_tag

Definition at line 146 of file [MemberModel.py](#).

#### 7.38.4.4 data

data

Definition at line 174 of file [MemberModel.py](#).

#### 7.38.4.5 E

E

Definition at line 138 of file [MemberModel.py](#).

#### 7.38.4.6 element\_array

element\_array

Definition at line 236 of file [MemberModel.py](#).

#### 7.38.4.7 geo\_transf\_ID

geo\_transf\_ID

Definition at line 141 of file [MemberModel.py](#).

#### 7.38.4.8 I\_rigid

I\_rigid

Definition at line 140 of file [MemberModel.py](#).

#### 7.38.4.9 Initialized

Initialized

Definition at line 148 of file [MemberModel.py](#).

#### 7.38.4.10 iNode\_ID

iNode\_ID

Definition at line 242 of file [MemberModel.py](#).

#### 7.38.4.11 jNode\_ID

jNode\_ID

Definition at line 243 of file [MemberModel.py](#).

#### 7.38.4.12 master\_node\_ID

master\_node\_ID

Definition at line 135 of file [MemberModel.py](#).

#### 7.38.4.13 mat\_ID

mat\_ID

Definition at line 142 of file [MemberModel.py](#).

#### 7.38.4.14 mid\_panel\_zone\_height

mid\_panel\_zone\_height

Definition at line 137 of file [MemberModel.py](#).

#### 7.38.4.15 mid\_panel\_zone\_width

mid\_panel\_zone\_width

Definition at line 136 of file [MemberModel.py](#).

#### 7.38.4.16 pin\_corners

pin\_corners

Definition at line 143 of file [MemberModel.py](#).

#### 7.38.4.17 spring\_ID

spring\_ID

Definition at line 158 of file [MemberModel.py](#).

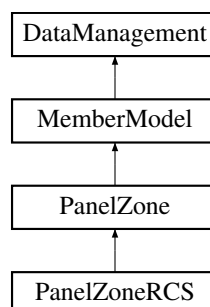
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.39 PanelZoneRCS Class Reference

WIP: Class that is the children of [PanelZone](#) and it's used for the panel zone in a RCS (RC column continous, Steel beam).

Inheritance diagram for PanelZoneRCS:



## Public Member Functions

- def `__init__` (self, int [master\\_node\\_ID](#), RCRectShape [col](#), SteelIShape [beam](#), int [geo\\_transf\\_ID](#), int [mat\\_ID](#), rigid=RIGID)

*Constructor of the class.*

## Public Attributes

- [beam](#)
- [beam\\_section\\_name\\_tag](#)
- [col](#)
- [col\\_section\\_name\\_tag](#)

### 7.39.1 Detailed Description

WIP: Class that is the children of [PanelZone](#) and it's used for the panel zone in a RCS (RC column continous, Steel beam).

Note that the corners are not pinned (do it manually).

#### Parameters

<a href="#">PanelZone</a>	Parent class.
---------------------------	---------------

Definition at line 306 of file [MemberModel.py](#).

### 7.39.2 Constructor & Destructor Documentation

#### 7.39.2.1 `__init__()`

```
def __init__ (
    self,
    int master_node_ID,
    RCRectShape col,
    SteelIShape beam,
    int geo_transf_ID,
    int mat_ID,
    rigid = RIGID )
```

Constructor of the class.

#### Parameters

<a href="#">master_node_ID</a>	(int): ID of the master node (central top node that should be a grid node).
<a href="#">col</a>	(RCRectShape): RCRectShape column section object.
<a href="#">beam</a>	(SteelIShape): SteelIShape beam section object.
<a href="#">geo_transf_ID</a>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<a href="#">mat_ID</a>	(int): ID of the material model for the panel zone spring.
<a href="#">rigid</a> <small>Generated by Doxygen</small>	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from [PanelZone](#).

Definition at line 313 of file [MemberModel.py](#).

```
00313     def __init__(self, master_node_ID: int, col: RCRectShape, beam: SteelIShape, geo_transf_ID: int,
00314                  mat_ID: int, rigid = RIGID):
00315         """
00316         Constructor of the class.
00317
00318         @param master_node_ID (int): ID of the master node (central top node that should be a grid
00319         node).
00320         @param col (RCRectShape): RCRectShape column section object.
00321         @param beam (SteelIShape): SteelIShape beam section object.
00322         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00323         documentation).
00324         @param mat_ID (int): ID of the material model for the panel zone spring.
00325         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
00326         element but enough small to avoid convergence problem. Defaults to RIGID.
00327         """
00328         self.col = deepcopy(col)
00329         self.beam = deepcopy(beam)
00330         super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, beam.E, max(col.A, beam.A)*rigid,
00331                          max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID, False)
00332
00333         self.col_section_name_tag = col.name_tag
00334         self.beam_section_name_tag = beam.name_tag
00335         self.UpdateStoredData()
```

## 7.39.3 Member Data Documentation

### 7.39.3.1 beam

beam

Definition at line 326 of file [MemberModel.py](#).

### 7.39.3.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 330 of file [MemberModel.py](#).

### 7.39.3.3 col

col

Definition at line 325 of file [MemberModel.py](#).



### 7.39.3.4 col\_section\_name\_tag

col\_section\_name\_tag

Definition at line 329 of file [MemberModel.py](#).

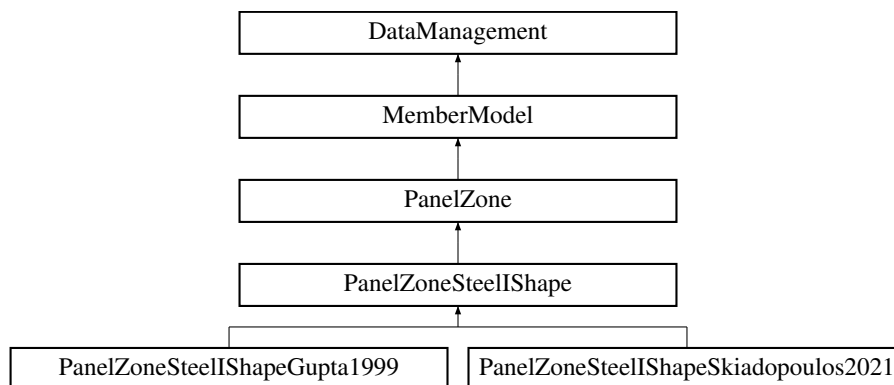
The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.40 PanelZoneSteelShape Class Reference

Class that is the children of [PanelZone](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for PanelZoneSteelShape:



### Public Member Functions

- def `__init__` (self, int [master\\_node\\_ID](#), SteelShape [col](#), SteelShape [beam](#), int [geo\\_transf\\_ID](#), int [mat\\_ID](#), rigid=RIGID)  
*Constructor of the class.*

### Public Attributes

- [beam](#)
- [beam\\_section\\_name\\_tag](#)
- [col](#)
- [col\\_section\\_name\\_tag](#)

### 7.40.1 Detailed Description

Class that is the children of [PanelZone](#) and combine the class SteelShape (section) to retrieve the information needed.

## Parameters

<a href="#">PanelZone</a>	Parent class.
---------------------------	---------------

Definition at line 279 of file [MemberModel.py](#).

## 7.40.2 Constructor & Destructor Documentation

### 7.40.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int master_node_ID,
    SteelIShape col,
    SteelIShape beam,
    int geo_transf_ID,
    int mat_ID,
    rigid = RIGID )
```

Constructor of the class.

## Parameters

<i>master_node_ID</i>	(int): ID of the master node (central top node that should be a grid node).
<i>col</i>	(SteelIShape): SteelIShape column section object.
<i>beam</i>	(SteelIShape): SteelIShape beam section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>mat_ID</i>	(int): ID of the material model for the panel zone spring.
<i>rigid</i>	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from [PanelZone](#).

Reimplemented in [PanelZoneSteelIShapeGupta1999](#), and [PanelZoneSteelIShapeSkiadopoulos2021](#).

Definition at line 285 of file [MemberModel.py](#).

```
00285     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
00286                 mat_ID: int, rigid = RIGID):
00287         """
00288             Constructor of the class.
00289             @param master_node_ID (int): ID of the master node (central top node that should be a grid
00290             node).
00291             @param col (SteelIShape): SteelIShape column section object.
00292             @param beam (SteelIShape): SteelIShape beam section object.
00293             @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00294             documentation).
00295             @param mat_ID (int): ID of the material model for the panel zone spring.
00296             @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
00297             element
00298             but enough small to avoid convergence problem. Defaults to RIGID.
00299             """
00300             self.col = deepcopy(col)
00301             self.beam = deepcopy(beam)
```

```
00299         super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, col.E, max(col.A, beam.A)*rigid,
00300                           max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID)
00301         self.col_section_name_tag = col.name_tag
00302         self.beam_section_name_tag = beam.name_tag
00303         self.UpdateStoredData()
00304
00305
```

## 7.40.3 Member Data Documentation

### 7.40.3.1 beam

beam

Definition at line 298 of file [MemberModel.py](#).

### 7.40.3.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 302 of file [MemberModel.py](#).

### 7.40.3.3 col

col

Definition at line 297 of file [MemberModel.py](#).

### 7.40.3.4 col\_section\_name\_tag

col\_section\_name\_tag

Definition at line 301 of file [MemberModel.py](#).

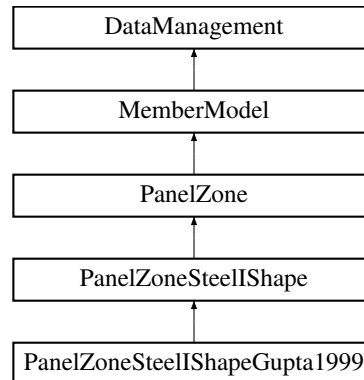
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.41 PanelZoneSteelIShapeGupta1999 Class Reference

Class that is the children of [PanelZoneSteelIShape](#) and automatically create the spring material model Gupta 1999 (ID = master\_node\_ID).

Inheritance diagram for PanelZoneSteelIShapeGupta1999:



### Public Member Functions

- `def __init__ (self, int master\_node\_ID, SteelIShape col, SteelIShape beam, int geo\_transf\_ID, t_dp=0, rigid=RIGID)`

*Constructor of the class.*

### Public Attributes

- [beam](#)
- [col](#)

#### 7.41.1 Detailed Description

Class that is the children of [PanelZoneSteelIShape](#) and automatically create the spring material model Gupta 1999 (ID = master\_node\_ID).

##### Parameters

<a href="#">PanelZoneSteelIShape</a>	Parent class.
--------------------------------------	---------------

Definition at line 334 of file [MemberModel.py](#).

#### 7.41.2 Constructor & Destructor Documentation

### 7.41.2.1 `__init__()`

```
def __init__ (
    self,
    int master_node_ID,
    SteelIShape col,
    SteelIShape beam,
    int geo_transf_ID,
    t_dp = 0,
    rigid = RIGID )
```

Constructor of the class.

#### Parameters

<i>master_node_ID</i>	(int): ID of the master node (central top node that should be a grid node).
<i>col</i>	(SteelIShape): SteelIShape column section object.
<i>beam</i>	(SteelIShape): SteelIShape beam section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.
<i>rigid</i>	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from [PanelZoneSteelShape](#).

Definition at line 340 of file [MemberModel.py](#).

```
00340     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
00341                 t_dp = 0, rigid=RIGID):
00342         """
00343         Constructor of the class.
00344         @param master_node_ID (int): ID of the master node (central top node that should be a grid
00345         node).
00346         @param col (SteelIShape): SteelIShape column section object.
00347         @param beam (SteelIShape): SteelIShape beam section object.
00348         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00349         documentation).
00350         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
00351         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
00352         element
00353         but enough small to avoid convergence problem. Defaults to RIGID.
00354         """
00355         self.col = deepcopy(col)
00356         self.beam = deepcopy(beam)
00357         mat_ID = master_node_ID
00358         pz_spring = Gupta1999SteelIShape(mat_ID, col, beam, t_dp)
00359         pz_spring.Hysteretic()
00360         super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)
```

## 7.41.3 Member Data Documentation

### 7.41.3.1 `beam`

`beam`

Definition at line 353 of file [MemberModel.py](#).

### 7.41.3.2 col

col

Definition at line 352 of file [MemberModel.py](#).

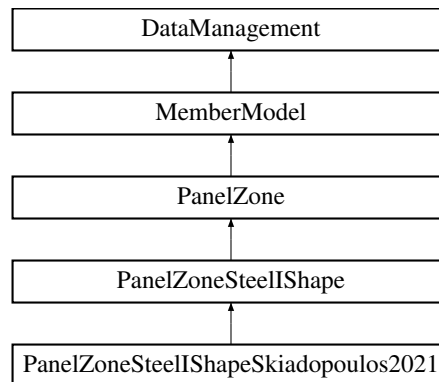
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.42 PanelZoneSteelShapeSkiadopoulos2021 Class Reference

Class that is the children of [PanelZoneSteelShape](#) and automatically create the spring material model Skiadopoulos 2021 (ID = master\_node\_ID).

Inheritance diagram for PanelZoneSteelShapeSkiadopoulos2021:



### Public Member Functions

- `def __init__(self, int master_node_ID, SteelShape col, SteelShape beam, int geo_transf_ID, t_dp=0, rigid=RIGID)`

*Constructor of the class.*

### Public Attributes

- `beam`
- `col`

### 7.42.1 Detailed Description

Class that is the children of [PanelZoneSteelShape](#) and automatically create the spring material model Skiadopoulos 2021 (ID = master\_node\_ID).

## Parameters

<a href="#">PanelZoneSteelShape</a>	Parent class.
-------------------------------------	---------------

Definition at line 361 of file [MemberModel.py](#).

## 7.42.2 Constructor & Destructor Documentation

### 7.42.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int master_node_ID,
    SteelIShape col,
    SteelIShape beam,
    int geo_transf_ID,
    t_dp = 0,
    rigid = RIGID )
```

Constructor of the class.

## Parameters

<i>master_node_ID</i>	(int): ID of the master node (central top node that should be a grid node).
<i>col</i>	(SteelIShape): SteelIShape column section object.
<i>beam</i>	(SteelIShape): SteelIShape beam section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.
<i>rigid</i>	(float, optional): Parameter with a value enough big to assure rigidity of one element but enough small to avoid convergence problem. Defaults to RIGID.

Reimplemented from [PanelZoneSteelShape](#).

Definition at line 367 of file [MemberModel.py](#).

```
00367 def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
00368             t_dp = 0, rigid=RIGID):
00369     """
00370     Constructor of the class.
00371     @param master_node_ID (int): ID of the master node (central top node that should be a grid
00372     node).
00373     @param col (SteelIShape): SteelIShape column section object.
00374     @param beam (SteelIShape): SteelIShape beam section object.
00375     @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00376     documentation).
00377     @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
00378     @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
00379     element
00380     but enough small to avoid convergence problem. Defaults to RIGID.
00381     """
00382     self.col = deepcopy(col)
00383     self.beam = deepcopy(beam)
00384     mat_ID = master_node_ID
00385     pz_spring = Skiadopoulos2021SteelIShape(mat_ID, col, beam, t_dp)
00386     pz_spring.Hysteretic()
```

```

00385         super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)
00386
00387

```

### 7.42.3 Member Data Documentation

#### 7.42.3.1 beam

beam

Definition at line 380 of file [MemberModel.py](#).

#### 7.42.3.2 col

col

Definition at line 379 of file [MemberModel.py](#).

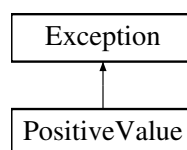
The documentation for this class was generated from the following file:

- [/media/carminе/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py](#)

## 7.43 PositiveValue Class Reference

Exception class for the "positive value (argument or result)" error.

Inheritance diagram for PositiveValue:



### 7.43.1 Detailed Description

Exception class for the "positive value (argument or result)" error.

Definition at line 21 of file [ErrorHandling.py](#).

The documentation for this class was generated from the following file:

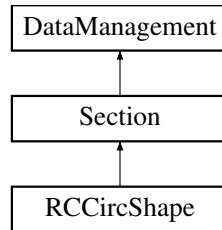
- [/media/carminе/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)



## 7.44 RCCircShape Class Reference

Class that stores functions, geometric and mechanical properties of RC circular shape profile.

Inheritance diagram for RCCircShape:



### Public Member Functions

- `def __init__ (self, b, L, e, fc, D_bars, int n_bars, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_vol=-1, Ec=-1)`  
*The constructor of the class.*
- `def ComputeEc (self)`  
*Compute Ec using the formula from Mander et Al.*
- `def Computel (self)`  
*Compute the moment of inertia of the circular section.*
- `def ComputeRhoVol (self)`  
*Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.*
- `def Relnit (self, rho_s_vol=-1, Ec=-1)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

### Public Attributes

- A
- Ac
- As
- Ay
- b
- bc
- cl\_bars
- cl\_hoops
- D\_bars
- D\_hoops
- data
- e
- Ec
- Es
- Ey
- fc

- [fs](#)
- [fy](#)
- [l](#)
- [L](#)
- [n\\_bars](#)
- [name\\_tag](#)
- [rho\\_bars](#)
- [rho\\_s\\_vol](#)
- [s](#)

### 7.44.1 Detailed Description

Class that stores functions, geometric and mechanical properties of RC circular shape profile.

Note that for the validity of the formulas, the hoops needs to be closed (with 135 degrees possibly).

#### Parameters

<a href="#">Section</a>	Parent abstract class.
-------------------------	------------------------

Definition at line [570](#) of file [Section.py](#).

### 7.44.2 Constructor & Destructor Documentation

#### 7.44.2.1 `__init__()`

```
def __init__ (
    self,
    b,
    L,
    e,
    fc,
    D_bars,
    int n_bars,
    fy,
    Ey,
    D_hoops,
    s,
    fs,
    Es,
    name_tag = "Not Defined",
    rho_s_vol = -1,
    Ec = -1 )
```

The constructor of the class.

#### Parameters

<i>b</i>	(float): Width of the section.
----------	--------------------------------

## Parameters

<i>L</i>	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
<i>e</i>	(float): Concrete cover.
<i>fc</i>	(float): Unconfined concrete compressive strength (cylinder test).
<i>D_bars</i>	(float): Diameter of the vertical reinforcing bars.
<i>n_bars</i>	(int): Number of vertical reinforcing bars.
<i>fy</i>	(float): Yield stress for reinforcing bars.
<i>Ey</i>	(float): Young modulus for reinforcing bars.
<i>D_hoops</i>	(float): Diameter of the hoops.
<i>s</i>	(float): Vertical centerline spacing between hoops.
<i>fs</i>	(float): Yield stress for the hoops.
<i>Es</i>	(float): Young modulus for the hoops
<i>name_tag</i>	(str, optional): A nametag for the section. Defaults to "Not Defined".
<i>rho_s_vol</i>	(float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core. Defaults to -1, e.g. computed according to Mander et Al. 1988.
<i>Ec</i>	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in <a href="#">init()</a> and <a href="#">ReInit()</a> .

## Exceptions

<i>NegativeValue</i>	b needs to be positive.
<i>NegativeValue</i>	L needs to be positive.
<i>NegativeValue</i>	e needs to be positive.
<i>PositiveValue</i>	fc needs to be negative.
<i>NegativeValue</i>	D_bars needs to be positive.
<i>NegativeValue</i>	n_bars needs to be a positive integer.
<i>NegativeValue</i>	fy needs to be positive.
<i>NegativeValue</i>	Ey needs to be positive.
<i>NegativeValue</i>	D_hoops needs to be positive.
<i>NegativeValue</i>	s needs to be positive.
<i>NegativeValue</i>	fs needs to be positive.
<i>NegativeValue</i>	Es needs to be positive.
<i>NegativeValue</i>	Ec needs to be positive if different from -1.
<i>InconsistentGeometry</i>	e should be smaller than half the depth and the width of the section.

Definition at line 577 of file [Section.py](#).

```

00577     def __init__(self, b, L, e, fc, D_bars, n_bars: int, fy, Ey, D_hoops, s, fs, Es, name_tag = "Not
    Defined", rho_s_vol = -1, Ec = -1):
00578         """
00579         The constructor of the class.
00580
00581         @param b (float): Width of the section.
00582         @param L (float): Effective length of the element associated with this section.
00583             If the panel zone is present, exclude its dimension.
00584         @param e (float): Concrete cover.
00585         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00586         @param D_bars (float): Diameter of the vertical reinforcing bars.
00587         @param n_bars (int): Number of vertical reinforcing bars.
00588         @param fy (float): Yield stress for reinforcing bars.
00589         @param Ey (float): Young modulus for reinforcing bars.
00590         @param D_hoops (float): Diameter of the hoops.
00591         @param s (float): Vertical centerline spacing between hoops.
00592         @param fs (float): Yield stress for the hoops.
00593         @param Es (float): Young modulus for the hoops
00594         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00595         @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the
    volume of confined concrete core.
```

```

00596         Defaults to -1, e.g. computed according to Mander et Al. 1988.
00597     @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in
__init__() and ReInit().
00598
00599     @exception NegativeValue: b needs to be positive.
00600     @exception NegativeValue: L needs to be positive.
00601     @exception NegativeValue: e needs to be positive.
00602     @exception PositiveValue: fc needs to be negative.
00603     @exception NegativeValue: D_bars needs to be positive.
00604     @exception NegativeValue: n_bars needs to be a positive integer.
00605     @exception NegativeValue: fy needs to be positive.
00606     @exception NegativeValue: Ey needs to be positive.
00607     @exception NegativeValue: D_hoops needs to be positive.
00608     @exception NegativeValue: s needs to be positive.
00609     @exception NegativeValue: fs needs to be positive.
00610     @exception NegativeValue: Es needs to be positive.
00611     @exception NegativeValue: Ec needs to be positive if different from -1.
00612     @exception InconsistentGeometry: e should be smaller than half the depth and the width of the
section.
00613     """
00614     # Check
00615     if b < 0: raise NegativeValue()
00616     if L < 0: raise NegativeValue()
00617     if e < 0: raise NegativeValue()
00618     if fc > 0: raise PositiveValue()
00619     if D_bars < 0: raise NegativeValue()
00620     if n_bars < 0: raise NegativeValue()
00621     if fy < 0: raise NegativeValue()
00622     if Ey < 0: raise NegativeValue()
00623     if D_hoops < 0: raise NegativeValue()
00624     if s < 0: raise NegativeValue()
00625     if fs < 0: raise NegativeValue()
00626     if Es < 0: raise NegativeValue()
00627     if Ec != -1 and Ec < 0: raise NegativeValue()
00628     if e > b/2: raise InconsistentGeometry()
00629
00630     # Arguments
00631     self.b = b
00632     self.L = L
00633     self.e = e
00634     self.fc = fc
00635     self.D_bars = D_bars
00636     self.n_bars = n_bars
00637     self.fy = fy
00638     self.Ey = Ey
00639     self.D_hoops = D_hoops
00640     self.s = s
00641     self.fs = fs
00642     self.Es = Es
00643     self.name_tag = name_tag
00644
00645     # Initialized the parameters that are dependent from others
00646     self.ReInit(rho_s_vol, Ec)
00647
00648

```

## 7.44.3 Member Function Documentation

### 7.44.3.1 ComputeEc()

```

def ComputeEc (
    self )

```

Compute Ec using the formula from Mander et Al.

1988.

**Returns**

float: Young modulus of concrete.

Definition at line 744 of file [Section.py](#).

```
00744     def ComputeEc(self):
00745         """
00746         Compute Ec using the formula from Mander et Al. 1988.
00747
00748         @returns float: Young modulus of concrete.
00749         """
00750
00751         return 5000.0 * math.sqrt(-self.fc/MPa_unit) * MPa_unit
00752
00753
```

**7.44.3.2 ComputeI()**

```
def ComputeI (
    self )
```

Compute the moment of inertia of the circular section.

**Returns**

float: Moment of inertia.

Definition at line 754 of file [Section.py](#).

```
00754     def ComputeI(self):
00755         """
00756         Compute the moment of inertia of the circular section.
00757
00758         @returns float: Moment of inertia.
00759         """
00760         return self.b**4*math.pi/64
00761
00762
```

**7.44.3.3 ComputeRhoVol()**

```
def ComputeRhoVol (
    self )
```

Compute the ratio of the volume of transverse confining steel to the volume of confined concrete core.

(according to Mander et Al. 1988).

**Returns**

float: Ratio.

Definition at line 731 of file [Section.py](#).

```
00731     def ComputeRhoVol(self):
00732         """
00733         Compute the ratio of the volume of transverse confining steel to the volume of confined
00734         concrete core.
00735         (according to Mander et Al. 1988).
00736
00737         @returns float: Ratio.
00738         """
00739         vol_s = self.As*math.pi*self.bc
00740         vol_c = math.pi/4*self.bc**2*self.s
00741
00742         return vol_s/vol_c
00743
```

### 7.44.3.4 ReInit()

```
def ReInit (
    self,
    rho_s_vol = -1,
    Ec = -1 )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>rho_s_vol</i>	(float, optional): Ratio of the volume of transverse confining steel to the volume of confined concrete core. Defaults to -1, e.g. computed according to Mander et Al. 1988.
<i>Ec</i>	(float): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.

Definition at line 649 of file [Section.py](#).

```
00649     def ReInit(self, rho_s_vol = -1, Ec = -1):
00650         """
00651         Implementation of the homonym abstract method.
00652         See parent class DataManagement for detailed information.
00653
00654         @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the
00655         volume of confined concrete core.
00656         Defaults to -1, e.g. computed according to Mander et Al. 1988.
00657         @param Ec (float): Young modulus for concrete. Defaults to -1, e.g. computed according to
00658         Mander et Al. 1988.
00659         """
00660         # Precompute some members
00661         self.cl_hoops = self.e + self.D_hoops/2.0 # centerline distance from the border of the extreme
00662         confining hoops
00663         self.cl_bars = self.e + self.D_bars/2.0 + self.D_hoops # centerline distance from the border
00664         of the corner bars
00665         self.bc = self.b - self.cl_hoops*2 # diameter of spiral (hoops) between bar centerline
00666         self.As = ComputeACircle(self.D_hoops)
00667
00668         # Arguments
00669         self.rho_s_vol = self.ComputeRhoVol() if rho_s_vol == -1 else rho_s_vol
00670         self.Ec = self.ComputeEc() if Ec == -1 else Ec
00671
00672         # Members
00673         self.A = ComputeACircle(self.b)
00674         self.Ac = ComputeACircle(self.bc)
00675         self.Ay = ComputeACircle(self.D_bars)
00676         self.rho_bars = ComputeRho(self.Ay, self.n_bars, self.A)
00677         self.I = self.ComputeI()
00678
00679         # Data storage for loading/saving
00680         self.UpdateStoredData()
```

### 7.44.3.5 ShowInfo()

```
def ShowInfo (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 710 of file [Section.py](#).

```

00710     def ShowInfo(self):
00711         """
00712         Implementation of the homonym abstract method.
00713         See parent class DataManagement for detailed information.
00714         """
00715         print("")
00716         print("Requested info for RC circular section of name tag = {}".format(self.name_tag))
00717         print("Width of the section b = {} mm".format(self.b/mm_unit))
00718         print("Concrete cover e = {} mm".format(self.e/mm_unit))
00719         print("Concrete area A = {} mm2".format(self.A/mm2_unit))
00720         print("Core concrete area Ac = {} mm2".format(self.Ac/mm2_unit))
00721         print("Unconfined concrete compressive strength fc = {} MPa".format(self.fc/MPa_unit))
00722         print("Young modulus for concrete Ec = {} GPa".format(self.Ec/GPa_unit))
00723         print("Diameter of the reinforcing bars DBars = {} mm and area of one bar Ay = {} mm2 with {}
bars".format(self.DBars/mm_unit, self.Ay/mm2_unit, self.nBars))
00724         print("Diameter of the hoops Dhoops = {} mm and area of one stirrup As = {}
mm2".format(self.Dhoops/mm_unit, self.As/mm2_unit))
00725         print("Ratio of area of longitudinal reinforcement to area of concrete section rhoBars = {}
".format(self.rhoBars))
00726         print("Ratio of the volume of transverse confining steel to the volume of confined concrete
core rho_s = {} ".format(self.rho_s_vol))
00727         print("Moment of inertia of the circular section I = {} mm4".format(self.I/mm4_unit))
00728         print("")
00729
00730

```

### 7.44.3.6 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 679 of file [Section.py](#).

```

00679     def UpdateStoredData(self):
00680         """
00681         Implementation of the homonym abstract method.
00682         See parent class DataManagement for detailed information.
00683         """
00684         self.data = [{"INFO_TYPE", "RCCircShape"}, # Tag for differentiating different data
00685             ["name_tag", self.name_tag],
00686             ["b", self.b],
00687             ["bc", self.bc],
00688             ["L", self.L],
00689             ["e", self.e],
00690             ["A", self.A],
00691             ["Ac", self.Ac],
00692             ["I", self.I],
00693             ["fc", self.fc],
00694             ["Ec", self.Ec],
00695             ["DBars", self.DBars],
00696             ["nBars", self.nBars],
00697             ["Ay", self.Ay],
00698             ["rhoBars", self.rhoBars],
00699             ["clBars", self.clBars],
00700             ["fy", self.fy],
00701             ["Ey", self.Ey],
00702             ["Dhoops", self.Dhoops],
00703             ["s", self.s],
00704             ["As", self.As],
00705             ["rho_s_vol", self.rho_s_vol],
00706             ["clhoops", self.clhoops],
00707             ["fs", self.fs],
00708             ["Es", self.Es]]
00709

```

## 7.44.4 Member Data Documentation

**7.44.4.1 A**

A

Definition at line [669](#) of file [Section.py](#).

**7.44.4.2 Ac**

Ac

Definition at line [670](#) of file [Section.py](#).

**7.44.4.3 As**

As

Definition at line [662](#) of file [Section.py](#).

**7.44.4.4 Ay**

Ay

Definition at line [671](#) of file [Section.py](#).

**7.44.4.5 b**

b

Definition at line [631](#) of file [Section.py](#).

**7.44.4.6 bc**

bc

Definition at line [661](#) of file [Section.py](#).



#### 7.44.4.7 clBars

clBars

Definition at line 660 of file [Section.py](#).

#### 7.44.4.8 clHoops

clHoops

Definition at line 659 of file [Section.py](#).

#### 7.44.4.9 DBars

DBars

Definition at line 635 of file [Section.py](#).

#### 7.44.4.10 DHoops

DHoops

Definition at line 639 of file [Section.py](#).

#### 7.44.4.11 data

data

Definition at line 684 of file [Section.py](#).

#### 7.44.4.12 e

e

Definition at line 633 of file [Section.py](#).

**7.44.4.13 Ec**

Ec

Definition at line 666 of file [Section.py](#).

**7.44.4.14 Es**

Es

Definition at line 642 of file [Section.py](#).

**7.44.4.15 Ey**

Ey

Definition at line 638 of file [Section.py](#).

**7.44.4.16 fc**

fc

Definition at line 634 of file [Section.py](#).

**7.44.4.17 fs**

fs

Definition at line 641 of file [Section.py](#).

**7.44.4.18 fy**

fy

Definition at line 637 of file [Section.py](#).

**7.44.4.19 I**

I

Definition at line 673 of file [Section.py](#).

**7.44.4.20 L**

L

Definition at line 632 of file [Section.py](#).

**7.44.4.21 nBars**

nBars

Definition at line 636 of file [Section.py](#).

**7.44.4.22 nameTag**

nameTag

Definition at line 643 of file [Section.py](#).

**7.44.4.23 rhoBars**

rhoBars

Definition at line 672 of file [Section.py](#).

**7.44.4.24 rhoSvol**

rhoSvol

Definition at line 665 of file [Section.py](#).

#### 7.44.4.25 s

s

Definition at line 640 of file [Section.py](#).

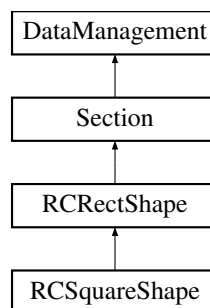
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/Section.py](#)

## 7.45 RCRectShape Class Reference

Class that stores functions, geometric and mechanical properties of RC rectangular shape profile.

Inheritance diagram for RCRectShape:



### Public Member Functions

- `def __init__ (self, b, d, L, e, fc, D_bars, np.ndarray bars_position_x, np.ndarray bars_ranges_position_y, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1)`  
*The constructor of the class.*
- `def ComputeA (self)`  
*Compute the area for a rectangular section.*
- `def ComputeAc (self)`  
*Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.*
- `def ComputeEc (self)`  
*Compute Ec using the formula from Mander et Al.*
- `def Computely (self)`  
*Compute the moment of inertia of the rectangular section with respect to the strong axis.*
- `def Computelz (self)`  
*Compute the moment of inertia of the rectangular section with respect to the weak axis.*
- `def ComputeNrBars (self)`  
*Compute the number of vertical bars in the array bars\_position\_x (note that this list of lists can have different list sizes).*
- `def Relnit (self, rho_s_x=-1, rho_s_y=-1, Ec=-1)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

## Public Attributes

- [A](#)
- [Ac](#)
- [As](#)
- [Ay](#)
- [b](#)
- [bars\\_position\\_x](#)
- [bars\\_ranges\\_position\\_y](#)
- [bc](#)
- [cl\\_bars](#)
- [cl\\_hoops](#)
- [d](#)
- [D\\_bars](#)
- [D\\_hoops](#)
- [data](#)
- [dc](#)
- [e](#)
- [Ec](#)
- [Es](#)
- [Ey](#)
- [fc](#)
- [fs](#)
- [fy](#)
- [ly](#)
- [lz](#)
- [L](#)
- [name\\_tag](#)
- [nr\\_bars](#)
- [rho\\_bars](#)
- [rho\\_s\\_x](#)
- [rho\\_s\\_y](#)
- [s](#)

### 7.45.1 Detailed Description

Class that stores functions, geometric and mechanical properties of RC rectangular shape profile.

Note that for the validity of the formulas, at least one bar per corner and at least one hoop closed (with 135 degrees possibly).

#### Parameters

<a href="#">Section</a>	Parent abstract class.
-------------------------	------------------------

Definition at line 264 of file [Section.py](#).

### 7.45.2 Constructor & Destructor Documentation

### 7.45.2.1 `__init__()`

```
def __init__ (
    self,
    b,
    d,
    L,
    e,
    fc,
    D_bars,
    np.ndarray bars_position_x,
    np.ndarray bars_ranges_position_y,
    fy,
    Ey,
    D_hoops,
    s,
    fs,
    Es,
    name_tag = "Not Defined",
    rho_s_x = -1,
    rho_s_y = -1,
    Ec = -1 )
```

The constructor of the class.

#### Parameters

<i>b</i>	(float): Width of the section.
<i>d</i>	(float): Depth of the section.
<i>L</i>	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
<i>e</i>	(float): Concrete cover.
<i>fc</i>	(float): Unconfined concrete compressive strength (cylinder test).
<i>D_bars</i>	(float): Diameter of the reinforcing bars.
<i>bars_position_x</i>	(np.ndarray): Array with a range of aligned vertical reinforcing bars for each row in x direction. Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned). Starting from the left to right, from the top range to the bottom one. The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
<i>bars_ranges_position_y</i>	(np.ndarray): Array of dimension 1 with the position or spacing in y of the ranges in bars_position_x. Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction. Starting from the top range to the bottom one.
<i>fy</i>	(float): Yield stress for reinforcing bars.
<i>Ey</i>	(float): Young modulus for reinforcing bars.
<i>D_hoops</i>	(float): Diameter of the hoops.
<i>s</i>	(float): Centerline distance for the hoops.
<i>fs</i>	(float): Yield stress for the hoops.
<i>Es</i>	(float): Young modulus for the hoops
<i>name_tag</i>	(str, optional): A nametag for the section. Defaults to "Not Defined".
<i>rho_s_x</i>	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction. Defaults to -1, e.g. computed in <code>init()</code> and <code>Relnit()</code> assuming one range of hoops.
<i>rho_s_y</i>	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction. Defaults to -1, e.g. computed in <code>init()</code> and <code>Relnit()</code> assuming one range of hoops.

## Parameters

<i>Ec</i>	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in <code>init()</code> and <code>Relnit()</code> .
-----------	---

## Exceptions

<i>NegativeValue</i>	b needs to be positive.
<i>NegativeValue</i>	d needs to be positive.
<i>NegativeValue</i>	L needs to be positive.
<i>NegativeValue</i>	e needs to be positive.
<i>PositiveValue</i>	fc needs to be negative.
<i>NegativeValue</i>	D_bars needs to be positive.
<i>NegativeValue</i>	fy needs to be positive.
<i>NegativeValue</i>	Ey needs to be positive.
<i>NegativeValue</i>	D_hoops needs to be positive.
<i>NegativeValue</i>	s needs to be positive.
<i>NegativeValue</i>	fs needs to be positive.
<i>NegativeValue</i>	Es needs to be positive.
<i>NegativeValue</i>	rho_s_x needs to be positive if different from -1.
<i>NegativeValue</i>	rho_s_y needs to be positive if different from -1.
<i>NegativeValue</i>	Ec needs to be positive if different from -1.
<i>WrongDimension</i>	Number of lists in the list bars_position_x needs to be the same of the length of bars_ranges_position_y - 1.
<i>InconsistentGeometry</i>	The sum of the distances for each list in bars_position_x should be equal to the section's width (tol = 5 mm).
<i>InconsistentGeometry</i>	The sum of the distances in bars_ranges_position_y should be equal to the section's depth (tol = 5 mm).
<i>InconsistentGeometry</i>	e should be smaller than half the depth and the width of the section.

Reimplemented in [RCSquareShape](#).

Definition at line 271 of file [Section.py](#).

```

00272         D_hoops, s, fs, Es, name_tag = "Not Defined", rho_s_x = -1, rho_s_y = -1, Ec = -1):
00273         """
00274         The constructor of the class.
00275
00276         @param b (float): Width of the section.
00277         @param d (float): Depth of the section.
00278         @param L (float): Effective length of the element associated with this section.
00279         If the panel zone is present, exclude its dimension.
00280         @param e (float): Concrete cover.
00281         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00282         @param D_bars (float): Diameter of the reinforcing bars.
00283         @param bars_position_x (np.ndarray): Array with a range of aligned vertical reinforcing bars
00284         for each row in x direction.
00285         Distances from border to bar centerline, bar to bar centerlines and
00286         finally bar centerline to border in the x direction (aligned).
00287         Starting from the left to right, from the top range to the bottom one.
00288         The number of bars for each range can vary; in this case, add this argument when defining
00289         the array " dtype = object".
00290         @param bars_ranges_position_y (np.ndarray): Array of dimension 1 with the position or spacing
00291         in y of the ranges in bars_position_x.
00292         Distances from border to range centerlines, range to range centerlines and
00293         finally range centerline to border in the y direction.
00294         Starting from the top range to the bottom one.
00295         @param fy (float): Yield stress for reinforcing bars.
00296         @param Ey (float): Young modulus for reinforcing bars.
00297         @param D_hoops (float): Diameter of the hoops.
00298         @param s (float): Centerline distance for the hoops.
00299         @param fs (float): Yield stress for the hoops.
00300         @param Es (float): Young modulus for the hoops

```

```

00298         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00299         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the x direction.
00300         Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00301         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the y direction.
00302         Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00303         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in
__init__() and ReInit().
00304
00305         @exception NegativeValue: b needs to be positive.
00306         @exception NegativeValue: d needs to be positive.
00307         @exception NegativeValue: L needs to be positive.
00308         @exception NegativeValue: e needs to be positive.
00309         @exception PositiveValue: fc needs to be negative.
00310         @exception NegativeValue: D_bars needs to be positive.
00311         @exception NegativeValue: fy needs to be positive.
00312         @exception NegativeValue: Ey needs to be positive.
00313         @exception NegativeValue: D_hoops needs to be positive.
00314         @exception NegativeValue: s needs to be positive.
00315         @exception NegativeValue: fs needs to be positive.
00316         @exception NegativeValue: Es needs to be positive.
00317         @exception NegativeValue: rho_s_x needs to be positive if different from -1.
00318         @exception NegativeValue: rho_s_y needs to be positive if different from -1.
00319         @exception NegativeValue: Ec needs to be positive if different from -1.
00320         @exception WrongDimension: Number of lists in the list bars_position_x needs to be the same of
the length of bars_ranges_position_y - 1.
00321         @exception InconsistentGeometry: The sum of the distances for each list in bars_position_x
should be equal to the section's width (tol = 5 mm).
00322         @exception InconsistentGeometry: The sum of the distances in bars_ranges_position_y should be
equal to the section's depth (tol = 5 mm).
00323         @exception InconsistentGeometry: e should be smaller than half the depth and the width of the
section.
00324         """
00325         # Check
00326         if b < 0: raise NegativeValue()
00327         if d < 0: raise NegativeValue()
00328         if L < 0: raise NegativeValue()
00329         if e < 0: raise NegativeValue()
00330         if fc > 0: raise PositiveValue()
00331         if D_bars < 0: raise NegativeValue()
00332         if fy < 0: raise NegativeValue()
00333         if Ey < 0: raise NegativeValue()
00334         if D_hoops < 0: raise NegativeValue()
00335         if s < 0: raise NegativeValue()
00336         if fs < 0: raise NegativeValue()
00337         if Es < 0: raise NegativeValue()
00338         if rho_s_x != -1 and rho_s_x < 0: raise NegativeValue()
00339         if rho_s_y != -1 and rho_s_y < 0: raise NegativeValue()
00340         if Ec != -1 and Ec < 0: raise NegativeValue()
00341         if np.size(bars_position_x) != np.size(bars_ranges_position_y)-1: raise WrongDimension()
00342         geometry_tol = 5*mm_unit
00343         for bars in bars_position_x:
00344             if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
00345         if abs(np.sum(bars_ranges_position_y)-d) > geometry_tol: raise InconsistentGeometry()
00346         if e > b/2 or e > d/2: raise InconsistentGeometry()
00347         warning_min_bars = "!!!!!! WARNING !!!!!!! The hypothesis of one bar per corner (aligned) is
not fulfilled."
00348         if len(bars_position_x) < 2:
00349             print(warning_min_bars)
00350         elif len(bars_position_x[0]) < 3 or len(bars_position_x[-1]) < 3:
00351             print(warning_min_bars)
00352
00353         # Arguments
00354         self.b = b
00355         self.d = d
00356         self.L = L
00357         self.e = e
00358         self.fc = fc
00359         self.D_bars = D_bars
00360         self.bars_position_x = deepcopy(bars_position_x)
00361         self.bars_ranges_position_y = copy(bars_ranges_position_y)
00362         self.fy = fy
00363         self.Ey = Ey
00364         self.D_hoops = D_hoops
00365         self.s = s
00366         self.fs = fs
00367         self.Es = Es
00368         self.name_tag = name_tag
00369
00370         # Initialized the parameters that are dependent from others
00371         self.ReInit(rho_s_x, rho_s_y, Ec)
00372
00373

```



## 7.45.3 Member Function Documentation

### 7.45.3.1 ComputeA()

```
def ComputeA (  
    self )
```

Compute the area for a rectangular section.

#### Returns

float: Total area.

Definition at line 495 of file [Section.py](#).

```
00495     def ComputeA(self):  
00496         """  
00497         Compute the area for a rectangular section.  
00498         @returns float: Total area.  
00499         """  
00500         return self.b * self.d  
00501  
00502  
00503
```

### 7.45.3.2 ComputeAc()

```
def ComputeAc (  
    self )
```

Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.

1988).

#### Returns

float: Confined area.

Definition at line 504 of file [Section.py](#).

```
00504     def ComputeAc(self):  
00505         """  
00506         Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.  
00507         1988).  
00508         @returns float: Confined area.  
00509         """  
00510         return self.bc * self.dc  
00511  
00512
```

### 7.45.3.3 ComputeEc()

```
def ComputeEc (
    self )
```

Compute Ec using the formula from Mander et Al.

1988.

#### Returns

float: Young modulus of concrete.

Definition at line 485 of file [Section.py](#).

```
00485     def ComputeEc(self):
00486         """
00487         Compute Ec using the formula from Mander et Al. 1988.
00488
00489         @returns float: Young modulus of concrete.
00490         """
00491
00492         return 5000.0 * math.sqrt(-self.fc/MPa_unit) * MPa_unit
00493
00494
```

### 7.45.3.4 Computely()

```
def Computely (
    self )
```

Compute the moment of inertia of the rectangular section with respect to the strong axis.

#### Returns

float: Moment of inertia (strong axis)

Definition at line 513 of file [Section.py](#).

```
00513     def Computely(self):
00514         """
00515         Compute the moment of inertia of the rectangular section with respect to the strong axis.
00516
00517         @returns float: Moment of inertia (strong axis)
00518         """
00519         return self.b * self.d**3 / 12.0
00520
00521
```

### 7.45.3.5 Computelz()

```
def Computelz (
    self )
```

Compute the moment of inertia of the rectangular section with respect to the weak axis.

#### Returns

float: Moment of inertia (weak axis)

Definition at line 522 of file [Section.py](#).

```
00522     def Computelz(self):
00523         """
00524         Compute the moment of inertia of the rectangular section with respect to the weak axis.
00525
00526         @returns float: Moment of inertia (weak axis)
00527         """
00528         return self.d * self.b**3 / 12.0
00529
00530
```

### 7.45.3.6 ComputeNrBars()

```
def ComputeNrBars (
    self )
```

Compute the number of vertical bars in the array `bars_position_x` (note that this list of lists can have different list sizes).

#### Returns

int: Number of vertical reinforcing bars.

Definition at line 472 of file [Section.py](#).

```
00472     def ComputeNrBars(self):
00473         """
00474         Compute the number of vertical bars in the array bars_position_x (note that this list of lists
00475         can have different list sizes).
00476
00477         @returns int: Number of vertical reinforcing bars.
00478         """
00479         nr_bars = 0
00480         for range in self.bars_position_x:
00481             nr_bars += np.size(range)-1
00482         return nr_bars
00483
00484
```

### 7.45.3.7 ReInit()

```
def ReInit (
    self,
    rho_s_x = -1,
    rho_s_y = -1,
    Ec = -1 )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

$\rho_{s_x}$	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction. Defaults to -1, e.g. computed assuming one range of hoops.
$\rho_{s_y}$	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction. Defaults to -1, e.g. computed assuming one range of hoops.
$E_c$	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed according to Mander et Al. 1988.

Definition at line 374 of file [Section.py](#).

```
00374     def ReInit(self, rho_s_x = -1, rho_s_y = -1, Ec = -1):
00375         """
00376         Implementation of the homonym abstract method.
00377         See parent class DataManagement for detailed information.
00378
00379         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
00380         concrete area in the x direction.
00381         Defaults to -1, e.g. computed assuming one range of hoops.
```

```

00381         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated
00382         concrete area in the y direction.
00383         Defaults to -1, e.g. computed assuming one range of hoops.
00384         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed
00385         according to Mander et Al. 1988.
00386         """
00387         # Precompute some members
00388         self.cl_hoops = self.e + self.D_hoops/2.0 # centerline distance from the border of the extreme
00389         confining hoops
00390         self.cl_bars = self.e + self.D_bars/2.0 + self.D_hoops # centerline distance from the border
00391         of the corner bars
00392         self.bc = self.b - self.cl_hoops*2
00393         self.dc = self.d - self.cl_hoops*2
00394         self.As = ComputeACircle(self.D_hoops)
00395
00396         # Arguments
00397         self.rho_s_x = 2.0*ComputeRho(self.As, 1, self.bc*self.s) if rho_s_x == -1 else rho_s_x
00398         self.rho_s_y = 2.0*ComputeRho(self.As, 1, self.dc*self.s) if rho_s_y == -1 else rho_s_y
00399         self.Ec = self.ComputeEc() if Ec == -1 else Ec
00400
00401         # Members
00402         self.nr_bars = self.ComputeNrBars()
00403         self.A = self.ComputeA()
00404         self.Ac = self.ComputeAc()
00405         self.Ay = ComputeACircle(self.D_bars)
00406         self.rho_bars = ComputeRho(self.Ay, self.nr_bars, self.A)
00407         self.Iy = self.ComputeIy()
00408         self.Iz = self.ComputeIz()
00409
00410         # Data storage for loading/saving
00411         self.UpdateStoredData()

```

### 7.45.3.8 ShowInfo()

```

def ShowInfo (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 448 of file `Section.py`.

```

00448     def ShowInfo(self):
00449         """
00450         Implementation of the homonym abstract method.
00451         See parent class DataManagement for detailed information.
00452         """
00453         print("")
00454         print("Requested info for RC rectangular section of name tag = {}".format(self.name_tag))
00455         print("Width of the section b = {} mm".format(self.b/mm_unit))
00456         print("Depth of the section d = {} mm".format(self.d/mm_unit))
00457         print("Concrete cover e = {} mm".format(self.e/mm_unit))
00458         print("Concrete area A = {} mm2".format(self.A/mm2_unit))
00459         print("Core concrete area Ac = {} mm2".format(self.Ac/mm2_unit))
00460         print("Unconfined concrete compressive strength fc = {} MPa".format(self.fc/MPa_unit))
00461         print("Young modulus for concrete Ec = {} GPa".format(self.Ec/GPa_unit))
00462         print("Diameter of the reinforcing bars D_bars = {} mm and area of one bar Ay = {} mm2 with {}
00463         bars".format(self.D_bars/mm_unit, self.Ay/mm2_unit, self.nr_bars))
00464         print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {}
00465         mm2".format(self.D_hoops/mm_unit, self.As/mm2_unit))
00466         print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bars =
00467         {}".format(self.rho_bars))
00468         print("Ratio of area of lateral reinforcement to lateral area of concrete section in x rho_s_x
00469         = {}".format(self.rho_s_x))
00470         print("Ratio of area of lateral reinforcement to lateral area of concrete section in y rho_s_y
00471         = {}".format(self.rho_s_y))
00472         print("Moment of inertia of the circular section (strong axis) Iy = {}
00473         mm4".format(self.Iy/mm4_unit))
00474         print("Moment of inertia of the circular section (weak axis) Iz = {}
00475         mm4".format(self.Iz/mm4_unit))
00476         print("")

```

### 7.45.3.9 UpdateStoredData()

```
def UpdateStoredData (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 410 of file [Section.py](#).

```
00410     def UpdateStoredData(self):
00411         """
00412         Implementation of the homonym abstract method.
00413         See parent class DataManagement for detailed information.
00414         """
00415         self.data = [{"INFO_TYPE", "RCRectShape"}, # Tag for differentiating different data
00416                     ["name_tag", self.name_tag],
00417                     ["b", self.b],
00418                     ["d", self.d],
00419                     ["bc", self.bc],
00420                     ["dc", self.dc],
00421                     ["L", self.L],
00422                     ["e", self.e],
00423                     ["A", self.A],
00424                     ["Ac", self.Ac],
00425                     ["Iy", self.Iy],
00426                     ["Iz", self.Iz],
00427                     ["fc", self.fc],
00428                     ["Ec", self.Ec],
00429                     ["D_bars", self.D_bars],
00430                     ["nr_bars", self.nr_bars],
00431                     ["Ay", self.Ay],
00432                     ["bars_position_x", self.bars_position_x],
00433                     ["bars_ranges_position_y", self.bars_ranges_position_y],
00434                     ["rho_bars", self.rho_bars],
00435                     ["cl_bars", self.cl_bars],
00436                     ["fy", self.fy],
00437                     ["Ey", self.Ey],
00438                     ["D_hoops", self.D_hoops],
00439                     ["s", self.s],
00440                     ["As", self.As],
00441                     ["rho_s_x", self.rho_s_x],
00442                     ["rho_s_y", self.rho_s_y],
00443                     ["cl_hoops", self.cl_hoops],
00444                     ["fs", self.fs],
00445                     ["Es", self.Es]]
00446
00447
```

## 7.45.4 Member Data Documentation

### 7.45.4.1 A

A

Definition at line 399 of file [Section.py](#).

### 7.45.4.2 Ac

Ac

Definition at line 400 of file [Section.py](#).

#### 7.45.4.3 As

As

Definition at line 390 of file [Section.py](#).

#### 7.45.4.4 Ay

Ay

Definition at line 401 of file [Section.py](#).

#### 7.45.4.5 b

b

Definition at line 354 of file [Section.py](#).

#### 7.45.4.6 bars\_position\_x

bars\_position\_x

Definition at line 360 of file [Section.py](#).

#### 7.45.4.7 bars\_ranges\_position\_y

bars\_ranges\_position\_y

Definition at line 361 of file [Section.py](#).

#### 7.45.4.8 bc

bc

Definition at line 388 of file [Section.py](#).

#### 7.45.4.9 clBars

clBars

Definition at line 387 of file [Section.py](#).

#### 7.45.4.10 clHoops

clHoops

Definition at line 386 of file [Section.py](#).

#### 7.45.4.11 d

d

Definition at line 355 of file [Section.py](#).

#### 7.45.4.12 DBars

DBars

Definition at line 359 of file [Section.py](#).

#### 7.45.4.13 DHoops

DHoops

Definition at line 364 of file [Section.py](#).

#### 7.45.4.14 data

data

Definition at line 415 of file [Section.py](#).

**7.45.4.15 dc**

dc

Definition at line 389 of file [Section.py](#).

**7.45.4.16 e**

e

Definition at line 357 of file [Section.py](#).

**7.45.4.17 Ec**

Ec

Definition at line 395 of file [Section.py](#).

**7.45.4.18 Es**

Es

Definition at line 367 of file [Section.py](#).

**7.45.4.19 Ey**

Ey

Definition at line 363 of file [Section.py](#).

**7.45.4.20 fc**

fc

Definition at line 358 of file [Section.py](#).



**7.45.4.21 fs**

fs

Definition at line 366 of file [Section.py](#).

**7.45.4.22 fy**

fy

Definition at line 362 of file [Section.py](#).

**7.45.4.23 ly**

ly

Definition at line 403 of file [Section.py](#).

**7.45.4.24 lz**

lz

Definition at line 404 of file [Section.py](#).

**7.45.4.25 L**

L

Definition at line 356 of file [Section.py](#).

**7.45.4.26 name\_tag**

name\_tag

Definition at line 368 of file [Section.py](#).

**7.45.4.27 nr\_bars**

`nr_bars`

Definition at line 398 of file [Section.py](#).

**7.45.4.28 rho\_bars**

`rho_bars`

Definition at line 402 of file [Section.py](#).

**7.45.4.29 rho\_s\_x**

`rho_s_x`

Definition at line 393 of file [Section.py](#).

**7.45.4.30 rho\_s\_y**

`rho_s_y`

Definition at line 394 of file [Section.py](#).

**7.45.4.31 s**

`s`

Definition at line 365 of file [Section.py](#).

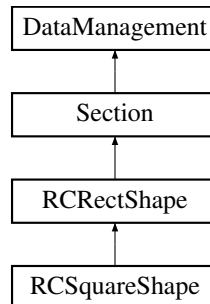
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py](#)

## 7.46 RCSquareShape Class Reference

Class that is the children of [RCRectShape](#) and cover the specific case of square RC sections.

Inheritance diagram for RCSquareShape:



### Public Member Functions

- `def __init__(self, b, L, e, fc, D_bars, np.ndarray bars_position_x, np.ndarray bars_ranges_position_y, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1)`  
*Constructor of the class.*

### Additional Inherited Members

#### 7.46.1 Detailed Description

Class that is the children of [RCRectShape](#) and cover the specific case of square RC sections.

#### Parameters

<a href="#">RCRectShape</a>	Parent class.
-----------------------------	---------------

Definition at line 531 of file [Section.py](#).

#### 7.46.2 Constructor & Destructor Documentation

##### 7.46.2.1 \_\_init\_\_()

```

def __init__(
    self,
    b,

```

```

        L,
        e,
        fc,
        D_bars,
        np.ndarray bars_position_x,
        np.ndarray bars_ranges_position_y,
        fy,
        Ey,
        D_hoops,
        s,
        fs,
        Es,
        name_tag = "Not Defined",
        rho_s_x = -1,
        rho_s_y = -1,
        Ec = -1 )

```

Constructor of the class.

It passes the arguments into the parent class to generate the specific case of a square RC section.

#### Parameters

<i>b</i>	(float): Width/depth of the section.
<i>L</i>	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
<i>e</i>	(float): Concrete cover.
<i>fc</i>	(float): Unconfined concrete compressive strength (cylinder test).
<i>D_bars</i>	(float): Diameter of the reinforcing bars.
<i>bars_position_x</i>	(np.ndarray): Distances from border to bar centerline, bar to bar centerlines and finally bar centerline to border in the x direction (aligned). Starting from the left to right, from the top range to the bottom one. The number of bars for each range can vary; in this case, add this argument when defining the array " dtype = object".
<i>bars_ranges_position_y</i>	(np.ndarray): Distances from border to range centerlines, range to range centerlines and finally range centerline to border in the y direction. Starting from the top range to the bottom one.
<i>fy</i>	(float): Yield stress for reinforcing bars.
<i>Ey</i>	(float): Young modulus for reinforcing bars.
<i>D_hoops</i>	(float): Diameter of the hoops.
<i>s</i>	(float): Vertical centerline spacing between hoops.
<i>fs</i>	(float): Yield stress for the hoops.
<i>Es</i>	(float): Young modulus for the hoops
<i>name_tag</i>	(str, optional): A nametag for the section. Defaults to "Not Defined".
<i>rho_s_x</i>	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the x direction. Defaults to -1, e.g. computed in <a href="#">init()</a> and <a href="#">Relnit()</a> assuming one range of hoops.
<i>rho_s_y</i>	(float, optional): Ratio of the transversal area of the hoops to the associated concrete area in the y direction. Defaults to -1, e.g. computed in <a href="#">init()</a> and <a href="#">Relnit()</a> assuming one range of hoops.
<i>Ec</i>	(float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in <a href="#">init()</a> and <a href="#">Relnit()</a> .

Reimplemented from [RCRectShape](#).

Definition at line 537 of file [Section.py](#).

```

00537     def __init__(self, b, L, e, fc, D_bars, bars_position_x: np.ndarray, bars_ranges_position_y:
np.ndarray, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1):
00538         """
00539         Constructor of the class. It passes the arguments into the parent class to generate the
specific case of a square RC section.
00540
00541         @param b (float): Width/depth of the section.
00542         @param L (float): Effective length of the element associated with this section.
00543         If the panel zone is present, exclude its dimension.
00544         @param e (float): Concrete cover.
00545         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00546         @param D_bars (float): Diameter of the reinforcing bars.
00547         @param bars_position_x (np.ndarray): Distances from border to bar centerline, bar to bar
centerlines and
00548         finally bar centerline to border in the x direction (aligned).
00549         Starting from the left to right, from the top range to the bottom one.
00550         The number of bars for each range can vary; in this case, add this argument when defining
the array " dtype = object".
00551         @param bars_ranges_position_y (np.ndarray): Distances from border to range centerlines, range
to range centerlines and
00552         finally range centerline to border in the y direction.
00553         Starting from the top range to the bottom one.
00554         @param fy (float): Yield stress for reinforcing bars.
00555         @param Ey (float): Young modulus for reinforcing bars.
00556         @param D_hoops (float): Diameter of the hoops.
00557         @param s (float): Vertical centerline spacing between hoops.
00558         @param fs (float): Yield stress for the hoops.
00559         @param Es (float): Young modulus for the hoops
00560         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00561         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the x direction.
00562         Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00563         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the y direction.
00564         Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00565         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in
__init__() and ReInit().
00566         """
00567         super().__init__(b, b, L, e, fc, D_bars, bars_position_x, bars_ranges_position_y, fy, Ey,
D_hoops, s, fs, Es, name_tag, rho_s_x, rho_s_y, Ec)
00568
00569

```

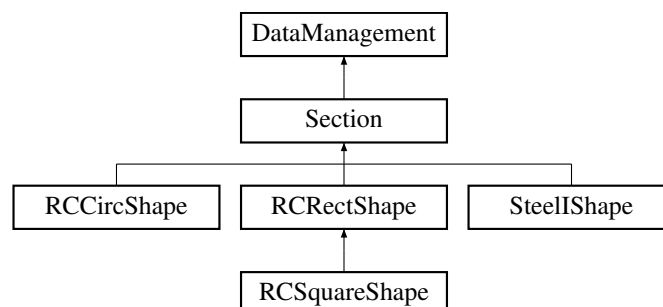
The documentation for this class was generated from the following file:

- [./media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py](#)

## 7.47 Section Class Reference

Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).

Inheritance diagram for Section:



### 7.47.1 Detailed Description

Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).

## Parameters

<i>DataManagement</i>	Parent abstract class.
-----------------------	------------------------

Definition at line 13 of file [Section.py](#).

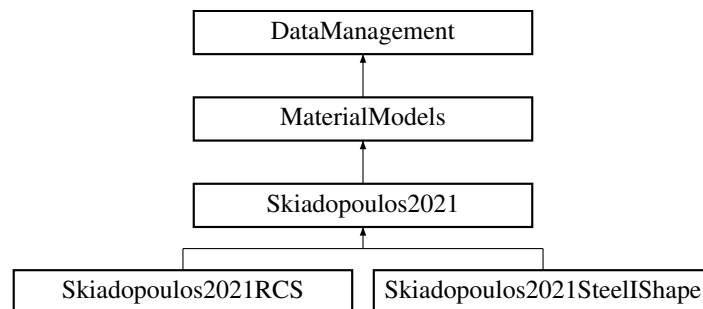
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/Section.py](#)

## 7.48 Skiadopoulos2021 Class Reference

Class that stores functions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

Inheritance diagram for Skiadopoulos2021:



### Public Member Functions

- `def __init__(self, int ID, d_c, bf_c, tf_c, l_c, d_b, tf_b, Fy, E, t_p, t_dp=0.0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0.0, dmg2=0.0, beta=0.0, safety_factor=False, t_fbp=0)`  
*Constructor of the class.*
- `def CheckApplicability(self)`  
*Implementation of the homonym abstract method.*
- `def Hysteretic(self)`  
*Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.*
- `def Relnit(self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo(self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData(self)`  
*Implementation of the homonym abstract method.*

## Public Attributes

- [a\\_s](#)
- [beam\\_section\\_name\\_tag](#)
- [beta](#)
- [bf\\_c](#)
- [Cf1](#)
- [Cf4](#)
- [Cf6](#)
- [col\\_section\\_name\\_tag](#)
- [Cw1](#)
- [Cw4](#)
- [Cw6](#)
- [d\\_b](#)
- [d\\_c](#)
- [data](#)
- [dmg1](#)
- [dmg2](#)
- [E](#)
- [Fy](#)
- [G](#)
- [Gamma\\_1](#)
- [Gamma\\_4](#)
- [Gamma\\_6](#)
- [I\\_c](#)
- [ID](#)
- [Initialized](#)
- [Kb](#)
- [Kbf](#)
- [Ke](#)
- [Kf](#)
- [Kf\\_Ke](#)
- [Ks](#)
- [Ksf](#)
- [M1](#)
- [M4](#)
- [M6](#)
- [pinchx](#)
- [pinchy](#)
- [Ry](#)
- [t\\_dp](#)
- [t\\_fbp](#)
- [t\\_p](#)
- [t\\_pz](#)
- [tf\\_b](#)
- [tf\\_c](#)
- [V1](#)
- [V4](#)
- [V6](#)

## Static Public Attributes

- list [Cf1\\_tests](#) = [0.035, 0.035, 0.033, 0.031, 0.018, 0.015, 0.013, 0.009, 0.009, 0.010, 0.010]
- list [Cf4\\_tests](#) = [0.145, 0.145, 0.123, 0.111, 0.069, 0.040, 0.040, 0.018, 0.010, 0.012, 0.012]
- list [Cf6\\_tests](#) = [0.165, 0.1650, 0.1400, 0.1275, 0.0800, 0.0500, 0.0500, 0.0180, 0.0140, 0.0120, 0.0120]
- list [Cw1\\_tests](#) = [0.96, 0.96, 0.955, 0.94, 0.93, 0.90, 0.89, 0.89, 0.88, 0.88, 0.88]
- list [Cw4\\_tests](#) = [1.145, 1.145, 1.140, 1.133, 1.120, 1.115, 1.115, 1.11, 1.10, 1.10, 1.10]
- list [Cw6\\_tests](#) = [1.205, 1.2050, 1.2000, 1.1925, 1.1740, 1.1730, 1.1720, 1.1690, 1.1670, 1.1650, 1.1650]
- list [Kf\\_Ke\\_tests](#) = [1.000, 0.153, 0.120, 0.090, 0.059, 0.031, 0.019, 0.009, 0.005, 0.004, 0.000]

### 7.48.1 Detailed Description

Class that stores functions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.

The material model is valid only if the column is continuous. For more information about the empirical model for the computation of the parameters, see Skiadopoulos et Al. 2021. The vectors that forms the matrix used to compute the material model parameters ([Kf\\_Ke\\_tests](#), [Cw1\\_tests](#), [Cf1\\_tests](#), [Cw4\\_tests](#), [Cf4\\_tests](#), [Cw6\\_tests](#), [Cf6\\_tests](#)) are used as global throughout the class to optimise the program (given the fact that is constant everytime).

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 780 of file [MaterialModels.py](#).

### 7.48.2 Constructor & Destructor Documentation

#### 7.48.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    d_c,
    bf_c,
    tf_c,
    I_c,
    d_b,
    tf_b,
    Fy,
    E,
    t_p,
    t_dp = 0.0,
    a_s = 0.03,
    pinchx = 0.25,
    pinchy = 0.75,
    dmg1 = 0.0,
    dmg2 = 0.0,
    beta = 0.0,
```



```
safety_factor = False,  
t_fbp = 0 )
```

Constructor of the class.

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>d_c</i>	(float): Column depth.
<i>bf_c</i>	(float): Column flange width.
<i>tf_c</i>	(float): Column flange thickness.
<i>I_c</i>	(float): Column moment of inertia (strong axis).
<i>d_b</i>	(float): Beam depth.
<i>tf_b</i>	(float): Beam flange thickness.
<i>Fy</i>	(float): Yield strength (if assume continous column, Fy of the web).
<i>E</i>	(float): Young modulus.
<i>t_p</i>	(float): Panel zone thickness.
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.0.
<i>a_s</i>	(float, optional): Strain hardening. Defaults to 0.03.
<i>pinchx</i>	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
<i>pinchy</i>	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
<i>dmg1</i>	(float, optional): Damage due to ductility: D1( $\mu$ -1). Defaults to 0.0.
<i>dmg2</i>	(float, optional): Damage due to energy: D2( $E_{ii}/E_{ult}$ ). Defaults to 0.0.
<i>beta</i>	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, $\mu$ -beta. Defaults to 0.0.
<i>safety_factor</i>	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
<i>t_fbp</i>	(float, optional): Thickness of the face bearing plate (if present). Defaults to 0.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	d_c needs to be positive.
<i>NegativeValue</i>	bf_c needs to be positive.
<i>NegativeValue</i>	tf_c needs to be positive.
<i>NegativeValue</i>	d_b needs to be positive.
<i>NegativeValue</i>	tf_b needs to be positive.
<i>NegativeValue</i>	Fy needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	t_p needs to be positive.
<i>NegativeValue</i>	a_s needs to be positive.

Reimplemented in [Skiadopoulos2021RCS](#), and [Skiadopoulos2021SteelShape](#).

Definition at line 808 of file [MaterialModels.py](#).

```

00809         t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
        safety_factor = False, t_fbp = 0):
00810     """
00811         Constructor of the class.
00812
00813         @param ID (int): Unique material model ID.
00814         @param d_c (float): Column depth.
00815         @param bf_c (float): Column flange width.
00816         @param tf_c (float): Column flange thickness.
00817         @param I_c (float): Column moment of inertia (strong axis).
00818         @param d_b (float): Beam depth.
00819         @param tf_b (float): Beam flange thickness.
00820         @param Fy (float): Yield strength (if assume continous column, Fy of the web).
00821         @param E (float): Young modulus.
00822         @param t_p (float): Panel zone thickness.
```

```

00823         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00824         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00825         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
Defaults to 0.25
00826         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
Defaults to 0.75
00827         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00828         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00829         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
on ductility, mu-beta. Defaults to 0.0.
00830         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
are used (not test results). Defaults to False.
00831         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to
0.
00832
00833         @exception NegativeValue: ID needs to be a positive integer.
00834         @exception NegativeValue: d_c needs to be positive.
00835         @exception NegativeValue: bf_c needs to be positive.
00836         @exception NegativeValue: tf_c needs to be positive.
00837         @exception NegativeValue: d_b needs to be positive.
00838         @exception NegativeValue: tf_b needs to be positive.
00839         @exception NegativeValue: Fy needs to be positive.
00840         @exception NegativeValue: E needs to be positive.
00841         @exception NegativeValue: t_p needs to be positive.
00842         @exception NegativeValue: a_s needs to be positive.
00843         """
00844         # Check
00845         if ID < 1: raise NegativeValue()
00846         if d_c < 0: raise NegativeValue()
00847         if bf_c < 0: raise NegativeValue()
00848         if tf_c < 0: raise NegativeValue()
00849         if d_b < 0: raise NegativeValue()
00850         if tf_b < 0: raise NegativeValue()
00851         if Fy < 0: raise NegativeValue()
00852         if E < 0: raise NegativeValue()
00853         if t_p < 0: raise NegativeValue()
00854         if a_s < 0: raise NegativeValue()
00855         if t_fbp < 0: raise NegativeValue()
00856
00857         # Arguments
00858         self.ID = ID
00859         self.d_c = d_c
00860         self.bf_c = bf_c
00861         self.tf_c = tf_c
00862         self.I_c = I_c
00863         self.d_b = d_b
00864         self.tf_b = tf_b
00865         self.Fy = Fy
00866         self.E = E
00867         self.t_p = t_p
00868         self.t_dp = t_dp
00869         self.a_s = a_s
00870         self.pinchx = pinchx
00871         self.pinchy = pinchy
00872         self.dmg1 = dmg1
00873         self.dmg2 = dmg2
00874         self.beta = beta
00875         if safety_factor:
00876             self.Ry = 1.2
00877         else:
00878             self.Ry = 1.0
00879         self.t_fbp = t_fbp
00880
00881         # Initialized the parameters that are dependent from others
00882         self.beam_section_name_tag = "None"
00883         self.col_section_name_tag = "None"
00884         self.Initialized = False
00885         self.ReInit()
00886
00887

```

## 7.48.3 Member Function Documentation

### 7.48.3.1 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 1025 of file [MaterialModels.py](#).

```
01025     def CheckApplicability(self):
01026         """
01027         Implementation of the homonym abstract method.
01028         See parent class DataManagement for detailed information.
01029         """
01030         Check = True
01031         # No checks
01032         if not Check:
01033             print("The validity of the equations is not fullfilled.")
01034             print("!!!!!! WARNING !!!!!!! Check material model of Skiadopoulos 2021, ID=", self.ID)
01035             print("")
01036
01037
```

### 7.48.3.2 Hysteretic()

```
def Hysteretic (
    self )
```

Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.

See `_Hysteretic` function for more information.

Definition at line 1038 of file [MaterialModels.py](#).

```
01038     def Hysteretic(self):
01039         """
01040         Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.
01041         See _Hysteretic function for more information.
01042         """
01043         _Hysteretic(self.ID, self.M1, self.Gamma_1, self.M4, self.Gamma_4, self.M6, self.Gamma_6,
01044                     self.pinchx, self.pinchy, self.dmg1, self.dmg2, self.beta)
01045         self.Initialized = True
01046         self.UpdateStoredData()
01047
01048
```

### 7.48.3.3 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 889 of file [MaterialModels.py](#).

```
00889     def ReInit(self):
00890         """
00891         Implementation of the homonym abstract method.
00892         See parent class DataManagement for detailed information.
00893         """
00894         # Check applicability
00895         self.CheckApplicability()
00896
00897         # Memebers
```

```

00898         if self.beam_section_name_tag != "None": self.beam_section_name_tag =
self.beam_section_name_tag + " (modified)"
00899         if self.col_section_name_tag != "None": self.col_section_name_tag = self.col_section_name_tag
+ " (modified)"
00900         self.t_pz = self.t_p + self.t_dp
00901         self.G = self.E/(2.0 * (1.0 + 0.30)) # Shear Modulus
00902
00903         # Refined computation of the parameters for the backbone curve for the panel zone spring
(Skiadopoulos et al. (2021))
00904         # Panel Zone Elastic Stiffness
00905         self.Ks = self.t_pz*(self.d_c-self.tf_c)*self.G
00906         self.Kb = 12.0*self.E*(self.I_c+self.t_dp*(self.d_c-2.0*self.tf_c)**3/12.0)/(self.d_b-0)**2
00907         self.Ke = self.Ks*self.Kb/(self.Ks+self.Kb)
00908
00909         # Column Flange Stiffness
00910         self.Ksf = 2.0*((self.tf_c+self.t_fbp)*self.bf_c*self.G)
00911         self.Kbf = 2.0*(12.0*self.E*self.bf_c*(self.tf_c**3+self.t_fbp**3)/12.0/(self.d_b-0)**2)
00912         self.Kf = self.Ksf*self.Kbf/(self.Ksf+self.Kbf)
00913
00914         # Kf/Ke Calculation for Panel Zone Categorization
00915         self.Kf_Ke = self.Kf/self.Ke
00916
00917         # Panel Zone Strength Coefficients (results from tests for a_w_eff and a_f_eff)
00918         self.Cw1 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cw1_tests)
00919         self.Cf1 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cf1_tests)
00920         self.Cw4 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cw4_tests)
00921         self.Cf4 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cf4_tests)
00922         self.Cw6 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cw6_tests)
00923         self.Cf6 = np.interp(self.Kf_Ke, Kf_Ke_tests, Cf6_tests)
00924
00925         # Panel Zone Model
00926         self.V1 = self.Fy*self.Ry/math.sqrt(3)*(self.Cw1*(self.d_c-self.tf_c)*self.t_pz +
self.Cf1*2*(self.bf_c-self.t_p)*self.tf_c)
00927         self.V4 = self.Fy*self.Ry/math.sqrt(3)*(self.Cw4*(self.d_c-self.tf_c)*self.t_pz +
self.Cf4*2*(self.bf_c-self.t_p)*self.tf_c)
00928         self.V6 = self.Fy*self.Ry/math.sqrt(3)*(self.Cw6*(self.d_c-self.tf_c)*self.t_pz +
self.Cf6*2*(self.bf_c-self.t_p)*self.tf_c)
00929
00930         self.M1 = self.V1*(self.d_b-self.tf_b)
00931         self.M4 = self.V4*(self.d_b-self.tf_b)
00932         self.M6 = self.V6*(self.d_b-self.tf_b)
00933
00934         self.Gamma_1 = self.V1/self.Ke
00935         self.Gamma_4 = 4*self.Gamma_1
00936         self.Gamma_6 = 6*self.Gamma_1
00937
00938         # Data storage for loading/saving
00939         self.UpdateStoredData()
00940
00941

```

#### 7.48.3.4 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

##### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 989 of file [MaterialModels.py](#).

```

00989     def ShowInfo(self, plot = False, block = False):
00990         """
00991         Implementation of the homonym abstract method.
00992         See parent class DataManagement for detailed information.
00993
00994         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00995         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00996         """
00997         print("")
00998         print("Requested info for Skiadopoulos 2021 material model Parameters, ID =
{}".format(self.ID))
00999         print("Sections associated, column: {}".format(self.col_section_name_tag))
01000         print("Sections associated, beam: {}".format(self.beam_section_name_tag))
01001         print("Gamma_1 = {} rad".format(self.Gamma_1))
01002         print("Gamma_4 = {} rad".format(self.Gamma_4))
01003         print("Gamma_6 = {} rad".format(self.Gamma_6))
01004         print("M1 = {} kNm".format(self.M1/kNm_unit))
01005         print("M4 = {} kNm".format(self.M4/kNm_unit))
01006         print("M6 = {} kNm".format(self.M6/kNm_unit))
01007         print("")
01008
01009         if plot:
01010             # Data for plotting
01011             x_axis = np.array([0.0, self.Gamma_1, self.Gamma_4, self.Gamma_6])
01012             y_axis = np.array([0.0, self.M1, self.M4, self.M6])/kNm_unit
01013
01014             fig, ax = plt.subplots()
01015             ax.plot(x_axis, y_axis, 'k-')
01016
01017             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
01018                   title='Skiadopoulos 2021 material model (ID={})'.format(self.ID))
01019             ax.grid()
01020
01021         if block:
01022             plt.show()
01023
01024

```

### 7.48.3.5 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 942 of file `MaterialModels.py`.

```

00942     def UpdateStoredData(self):
00943         """
00944         Implementation of the homonym abstract method.
00945         See parent class DataManagement for detailed information.
00946         """
00947         self.data = [{"INFO_TYPE", "Skiadopoulos2021"}, # Tag for differentiating different data
00948                     {"ID", self.ID},
00949                     {"beam_section_name_tag", self.beam_section_name_tag},
00950                     {"col_section_name_tag", self.col_section_name_tag},
00951                     {"d_c", self.d_c},
00952                     {"bf_c", self.bf_c},
00953                     {"tf_c", self.tf_c},
00954                     {"I_c", self.I_c},
00955                     {"d_b", self.d_b},
00956                     {"tf_b", self.tf_b},
00957                     {"Fy", self.Fy},
00958                     {"E", self.E},
00959                     {"G", self.G},
00960                     {"t_p", self.t_p},
00961                     {"t_dp", self.t_dp},
00962                     {"t_pz", self.t_pz},
00963                     {"a_s", self.a_s},
00964                     {"pinchx", self.pinchx},
00965                     {"pinchy", self.pinchy},
00966                     {"dmg1", self.dmg1},
00967                     {"dmg2", self.dmg2},

```

```
00968         ["beta", self.beta],
00969         ["Ry", self.Ry],
00970         ["Ks", self.Ks],
00971         ["Kb", self.Kb],
00972         ["Ke", self.Ke],
00973         ["Ksf", self.Ksf],
00974         ["Kbf", self.Kbf],
00975         ["Kf", self.Kf],
00976         ["Kf_Ke", self.Kf_Ke],
00977         ["V1", self.V1],
00978         ["V4", self.V4],
00979         ["V6", self.V6],
00980         ["M1", self.M1],
00981         ["M4", self.M4],
00982         ["M6", self.M6],
00983         ["Gamma_1", self.Gamma_1],
00984         ["Gamma_4", self.Gamma_4],
00985         ["Gamma_6", self.Gamma_6],
00986         ["Initialized", self.Initialized]]
00987
00988
```

## 7.48.4 Member Data Documentation

### 7.48.4.1 a\_s

a\_s

Definition at line 869 of file [MaterialModels.py](#).

### 7.48.4.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 882 of file [MaterialModels.py](#).

### 7.48.4.3 beta

beta

Definition at line 874 of file [MaterialModels.py](#).

### 7.48.4.4 bf\_c

bf\_c

Definition at line 860 of file [MaterialModels.py](#).

#### 7.48.4.5 Cf1

Cf1

Definition at line 919 of file [MaterialModels.py](#).

#### 7.48.4.6 Cf1\_tests

```
list Cf1_tests = [0.035, 0.035, 0.033, 0.031, 0.018, 0.015, 0.013, 0.009, 0.009, 0.010, 0.010]
[static]
```

Definition at line 797 of file [MaterialModels.py](#).

#### 7.48.4.7 Cf4

Cf4

Definition at line 921 of file [MaterialModels.py](#).

#### 7.48.4.8 Cf4\_tests

```
list Cf4_tests = [0.145, 0.145, 0.123, 0.111, 0.069, 0.040, 0.040, 0.018, 0.010, 0.012, 0.012]
[static]
```

Definition at line 801 of file [MaterialModels.py](#).

#### 7.48.4.9 Cf6

Cf6

Definition at line 923 of file [MaterialModels.py](#).

#### 7.48.4.10 Cf6\_tests

```
list Cf6_tests = [0.165, 0.1650, 0.1400, 0.1275, 0.0800, 0.0500, 0.0500, 0.0180, 0.0140, 0.↵
0120, 0.0120] [static]
```

Definition at line 805 of file [MaterialModels.py](#).



#### 7.48.4.11 col\_section\_name\_tag

col\_section\_name\_tag

Definition at line 883 of file [MaterialModels.py](#).

#### 7.48.4.12 Cw1

Cw1

Definition at line 918 of file [MaterialModels.py](#).

#### 7.48.4.13 Cw1\_tests

```
list Cw1_tests = [0.96, 0.96, 0.955, 0.94, 0.93, 0.90, 0.89, 0.89, 0.88, 0.88, 0.88] [static]
```

Definition at line 795 of file [MaterialModels.py](#).

#### 7.48.4.14 Cw4

Cw4

Definition at line 920 of file [MaterialModels.py](#).

#### 7.48.4.15 Cw4\_tests

```
list Cw4_tests = [1.145, 1.145, 1.140, 1.133, 1.120, 1.115, 1.115, 1.11, 1.10, 1.10, 1.10] [static]
```

Definition at line 799 of file [MaterialModels.py](#).

#### 7.48.4.16 Cw6

Cw6

Definition at line 922 of file [MaterialModels.py](#).

#### 7.48.4.17 Cw6\_tests

```
list Cw6_tests = [1.205, 1.2050, 1.2000, 1.1925, 1.1740, 1.1730, 1.1720, 1.1690, 1.1670, 1.1650, 1.1650] [static]
```

Definition at line 803 of file [MaterialModels.py](#).

#### 7.48.4.18 d\_b

d\_b

Definition at line 863 of file [MaterialModels.py](#).

#### 7.48.4.19 d\_c

d\_c

Definition at line 859 of file [MaterialModels.py](#).

#### 7.48.4.20 data

data

Definition at line 947 of file [MaterialModels.py](#).

#### 7.48.4.21 dmg1

dmg1

Definition at line 872 of file [MaterialModels.py](#).

#### 7.48.4.22 dmg2

dmg2

Definition at line 873 of file [MaterialModels.py](#).

**7.48.4.23 E**

E

Definition at line 866 of file [MaterialModels.py](#).

**7.48.4.24 Fy**

Fy

Definition at line 865 of file [MaterialModels.py](#).

**7.48.4.25 G**

G

Definition at line 901 of file [MaterialModels.py](#).

**7.48.4.26 Gamma\_1**

Gamma\_1

Definition at line 934 of file [MaterialModels.py](#).

**7.48.4.27 Gamma\_4**

Gamma\_4

Definition at line 935 of file [MaterialModels.py](#).

**7.48.4.28 Gamma\_6**

Gamma\_6

Definition at line 936 of file [MaterialModels.py](#).

**7.48.4.29 I\_c**

`I_c`

Definition at line [862](#) of file [MaterialModels.py](#).

**7.48.4.30 ID**

`ID`

Definition at line [858](#) of file [MaterialModels.py](#).

**7.48.4.31 Initialized**

`Initialized`

Definition at line [884](#) of file [MaterialModels.py](#).

**7.48.4.32 Kb**

`Kb`

Definition at line [906](#) of file [MaterialModels.py](#).

**7.48.4.33 Kbf**

`Kbf`

Definition at line [911](#) of file [MaterialModels.py](#).

**7.48.4.34 Ke**

`Ke`

Definition at line [907](#) of file [MaterialModels.py](#).

**7.48.4.35 Kf**

Kf

Definition at line 912 of file [MaterialModels.py](#).

**7.48.4.36 Kf\_Ke**

Kf\_Ke

Definition at line 915 of file [MaterialModels.py](#).

**7.48.4.37 Kf\_Ke\_tests**

```
list Kf_Ke_tests = [1.000, 0.153, 0.120, 0.090, 0.059, 0.031, 0.019, 0.009, 0.005, 0.004, 0.↵  
000] [static]
```

Definition at line 793 of file [MaterialModels.py](#).

**7.48.4.38 Ks**

Ks

Definition at line 905 of file [MaterialModels.py](#).

**7.48.4.39 Ksf**

Ksf

Definition at line 910 of file [MaterialModels.py](#).

**7.48.4.40 M1**

M1

Definition at line 930 of file [MaterialModels.py](#).

**7.48.4.41 M4**

M4

Definition at line 931 of file [MaterialModels.py](#).

**7.48.4.42 M6**

M6

Definition at line 932 of file [MaterialModels.py](#).

**7.48.4.43 pinchx**

pinchx

Definition at line 870 of file [MaterialModels.py](#).

**7.48.4.44 pinchy**

pinchy

Definition at line 871 of file [MaterialModels.py](#).

**7.48.4.45 Ry**

Ry

Definition at line 876 of file [MaterialModels.py](#).

**7.48.4.46 t\_dp**

t\_dp

Definition at line 868 of file [MaterialModels.py](#).

**7.48.4.47 t\_fbp**

t\_fbp

Definition at line 879 of file [MaterialModels.py](#).

**7.48.4.48 t\_p**

t\_p

Definition at line 867 of file [MaterialModels.py](#).

**7.48.4.49 t\_pz**

t\_pz

Definition at line 900 of file [MaterialModels.py](#).

**7.48.4.50 tf\_b**

tf\_b

Definition at line 864 of file [MaterialModels.py](#).

**7.48.4.51 tf\_c**

tf\_c

Definition at line 861 of file [MaterialModels.py](#).

**7.48.4.52 V1**

V1

Definition at line 926 of file [MaterialModels.py](#).

#### 7.48.4.53 V4

V4

Definition at line 927 of file [MaterialModels.py](#).

#### 7.48.4.54 V6

V6

Definition at line 928 of file [MaterialModels.py](#).

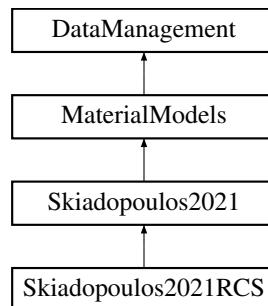
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.49 Skiadopoulos2021RCS Class Reference

WIP: Class that is the children of [Skiadopoulos2021](#) and it's used for the panel zone spring in a RCS (RC column continuous, Steel beam).

Inheritance diagram for Skiadopoulos2021RCS:



### Public Member Functions

- `def __init__(self, int ID, SteelShape beam, d_col, t_fbp=0, t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False)`

*Constructor of the class.*

### Public Attributes

- `beam`
- `beam_section_name_tag`

### Additional Inherited Members

#### 7.49.1 Detailed Description

WIP: Class that is the children of [Skiadopoulos2021](#) and it's used for the panel zone spring in a RCS (RC column continuous, Steel beam).



## Parameters

<a href="#">Skiadopoulos2021</a>	Parent class.
----------------------------------	---------------

Definition at line 1084 of file [MaterialModels.py](#).

## 7.49.2 Constructor & Destructor Documentation

### 7.49.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int ID,
    SteelIShape beam,
    d_col,
    t_fbp = 0,
    t_dp = 0,
    a_s = 0.03,
    pinchx = 0.25,
    pinchy = 0.75,
    dmg1 = 0,
    dmg2 = 0,
    beta = 0,
    safety_factor = False )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class SteelIShape. The copy of the section (beam) passed is stored in the member variable self.beam.

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>beam</i>	(SteelIShape): SteelIShape beam section object.
<i>d_col</i>	(float): Depth of the RC column (continous)
<i>t_fbp</i>	(float, optional): Thickness of the face bearing plate (if present). Defaults to 0.
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.0.
<i>a_s</i>	(float, optional): Strain hardening. Defaults to 0.03.
<i>pinchx</i>	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25
<i>pinchy</i>	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75
<i>dmg1</i>	(float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
<i>dmg2</i>	(float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
<i>beta</i>	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, mu-beta. Defaults to 0.0.
<i>safety_factor</i>	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.

Reimplemented from [Skiadopoulos2021](#).

Definition at line 1090 of file [MaterialModels.py](#).

```

01091         t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False):
01092         """
01093         Constructor of the class. It passes the arguments into the parent class to generate the
01094         combination of the parent class
01095         and the section class SteelIShape.
01096         The copy of the section (beam) passed is stored in the member variable self.beam.
01097         @param ID (int): Unique material model ID.
01098         @param beam (SteelIShape): SteelIShape beam section object.
01099         @param d_col (float): Depth of the RC column (continuous)
01100         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to
01101         0.
01102         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
01103         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
01104         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
01105         Defaults to 0.25
01106         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
01107         Defaults to 0.75
01108         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
01109         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
01110         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
01111         on ductility, mu-beta. Defaults to 0.0.
01112         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
01113         are used (not test results). Defaults to False.
01114         """
01115         self.beam = deepcopy(beam)
01116         super().__init__(ID, beam.d, beam.bf, beam.tf, beam.Iy, d_col, 0, beam.Fy_web, beam.E,
01117         beam.tw,
01118         t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta,
01119         safety_factor=safety_factor, t_fbp=t_fbp)
01120         self.beam_section_name_tag = beam.name_tag
01121         self.UpdateStoredData()
01122         self.beam_section_name_tag = beam.name_tag
01123         self.UpdateStoredData()
01124         self.beam_section_name_tag = beam.name_tag
01125         self.UpdateStoredData()
01126         self.beam_section_name_tag = beam.name_tag
01127         self.UpdateStoredData()

```

## 7.49.3 Member Data Documentation

### 7.49.3.1 beam

beam

Definition at line 1110 of file [MaterialModels.py](#).

### 7.49.3.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 1113 of file [MaterialModels.py](#).

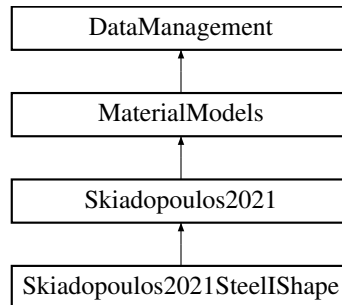
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[MaterialModels.py](#)

## 7.50 Skiadopoulos2021SteelShape Class Reference

Class that is the children of [Skiadopoulos2021](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for Skiadopoulos2021SteelShape:



### Public Member Functions

- `def __init__(self, int ID, SteelShape col, SteelShape beam, t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False, t_fbp=0)`

*Constructor of the class.*

### Public Attributes

- `beam`
- `beam_section_name_tag`
- `col`
- `col_section_name_tag`

### Additional Inherited Members

#### 7.50.1 Detailed Description

Class that is the children of [Skiadopoulos2021](#) and combine the class SteelShape (section) to retrieve the information needed.

#### Parameters

<a href="#">Skiadopoulos2021</a>	Parent class.
----------------------------------	---------------

Definition at line 1049 of file [MaterialModels.py](#).

## 7.50.2 Constructor & Destructor Documentation

### 7.50.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    SteelIShape col,
    SteelIShape beam,
    t_dp = 0,
    a_s = 0.03,
    pinchx = 0.25,
    pinchy = 0.75,
    dmg1 = 0,
    dmg2 = 0,
    beta = 0,
    safety_factor = False,
    t_fbp = 0 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class `SteelIShape`. The copy of the sections (`col` and `beam`) passed are stored in the member variable `self.col` and `self.beam`.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>col</i>	(SteelIShape): SteelIShape column section object.
<i>beam</i>	(SteelIShape): SteelIShape beam section object.
<i>t_dp</i>	(float, optional): Doubler plate thickness. Defaults to 0.0.
<i>a_s</i>	(float, optional): Strain hardening. Defaults to 0.03.
<i>pinchx</i>	(float, optional): Pinching factor for strain (or deformation) during reloading. Defaults to 0.25.
<i>pinchy</i>	(float, optional): Pinching factor for stress (or force) during reloading. Defaults to 0.75.
<i>dmg1</i>	(float, optional): Damage due to ductility: $D1(\mu-1)$ . Defaults to 0.0.
<i>dmg2</i>	(float, optional): Damage due to energy: $D2(E_{ii}/E_{ult})$ . Defaults to 0.0.
<i>beta</i>	(float, optional): Power used to determine the degraded unloading stiffness based on ductility, $\mu$ -beta. Defaults to 0.0.
<i>safety_factor</i>	(bool, optional): Safety factor used if standard mechanical parameters are used (not test results). Defaults to False.
<i>t_fbp</i>	(float, optional): Thickness of the face bearing plate (if present). Defaults to 0.

Reimplemented from [Skiadopoulos2021](#).

Definition at line 1055 of file [MaterialModels.py](#).

```
01056         t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False, t_fbp
    = 0):
01057         """
01058         Constructor of the class. It passes the arguments into the parent class to generate the
    combination of the parent class
01059         and the section class SteelIShape.
01060         The copy of the sections (col and beam) passed are stored in the member variable self.col and
    self.beam.
```

```

01061
01062     @param ID (int): Unique material model ID.
01063     @param col (SteelIShape): SteelIShape column section object.
01064     @param beam (SteelIShape): SteelIShape beam section object.
01065     @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
01066     @param a_s (float, optional): Strain hardening. Defaults to 0.03.
01067     @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
    Defaults to 0.25.
01068     @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
    Defaults to 0.75.
01069     @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
01070     @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
01071     @param beta (float, optional): Power used to determine the degraded unloading stiffness based
    on ductility, mu-beta. Defaults to 0.0.
01072     @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
    are used (not test results). Defaults to False.
01073     @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to
    0.
01074     """
01075     self.col = deepcopy(col)
01076     self.beam = deepcopy(beam)
01077     super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E,
    col.tw,
01078         t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta,
    safety_factor=safety_factor, t_fbp=t_fbp)
01079     self.beam_section_name_tag = beam.name_tag
01080     self.col_section_name_tag = col.name_tag
01081     self.UpdateStoredData()
01082
01083

```

### 7.50.3 Member Data Documentation

#### 7.50.3.1 beam

beam

Definition at line 1076 of file [MaterialModels.py](#).

#### 7.50.3.2 beam\_section\_name\_tag

beam\_section\_name\_tag

Definition at line 1079 of file [MaterialModels.py](#).

#### 7.50.3.3 col

col

Definition at line 1075 of file [MaterialModels.py](#).

### 7.50.3.4 col\_section\_name\_tag

col\_section\_name\_tag

Definition at line 1080 of file [MaterialModels.py](#).

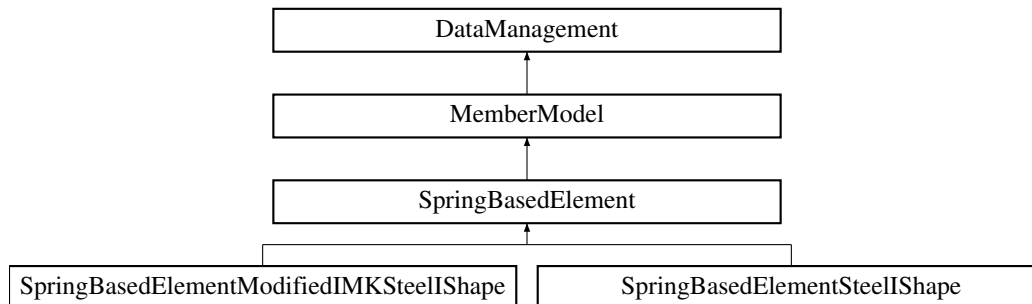
The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[MaterialModels.py](#)

## 7.51 SpringBasedElement Class Reference

Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

Inheritance diagram for SpringBasedElement:



### Public Member Functions

- `def __init__ (self, int iNode_ID, int jNode_ID, A, E, ly_mod, int geo_transf_ID, mat_ID_i=-1, mat_ID_j=-1, ele_ID=-1)`  
*Constructor of the class.*
- `def CreateMember (self)`  
*Method that initialises the member by calling the OpenSeesPy commands through various functions.*
- `def Record (self, str spring_or_element, str name_txt, str data_dir, force_rec=True, def_rec=True, time_rec=True)`  
*Implementation of the homonym abstract method.*
- `def RecordNodeDef (self, str name_txt, str data_dir, time_rec=True)`  
*Implementation of the homonym abstract method.*
- `def ReInit (self, ele_ID=-1)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo (self, plot=False, block=False)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData (self)`  
*Implementation of the homonym abstract method.*

## Public Attributes

- [A](#)
- [data](#)
- [E](#)
- [ele\\_orientation](#)
- [element\\_array](#)
- [element\\_ID](#)
- [geo\\_transf\\_ID](#)
- [Initialized](#)
- [iNode\\_ID](#)
- [iNode\\_ID\\_spring](#)
- [iSpring\\_ID](#)
- [Iy\\_mod](#)
- [jNode\\_ID](#)
- [jNode\\_ID\\_spring](#)
- [jSpring\\_ID](#)
- [mat\\_ID\\_i](#)
- [mat\\_ID\\_j](#)
- [section\\_name\\_tag](#)

### 7.51.1 Detailed Description

Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

#### Parameters

<a href="#">MemberModel</a>	Parent abstract class.
-----------------------------	------------------------

Definition at line 659 of file [MemberModel.py](#).

### 7.51.2 Constructor & Destructor Documentation

#### 7.51.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    A,
    E,
    Iy_mod,
    int geo_transf_ID,
    mat_ID_i = -1,
    mat_ID_j = -1,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>A</i>	(float): Area of the member.
<i>E</i>	(float): Young modulus.
<i>Iy_mod</i>	(float): Second moment of inertia (strong axis).
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>mat_ID_i</i>	(int, optional): ID of the material model for the spring in the node i (if present). Defaults to -1.
<i>mat_ID_j</i>	(int, optional): ID of the material model for the spring in the node j (if present). Defaults to -1.
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	A needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	Iy_mod needs to be positive.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer, if different from -1.
<i>NegativeValue</i>	ID needs to be a positive integer, if different from -1.
<i>NameError</i>	at least one spring needs to be defined.
<i>NegativeValue</i>	ID needs to be a positive integer, if different from -1.

Reimplemented in [SpringBasedElementSteelShape](#), and [SpringBasedElementModifiedIMKSteelShape](#).

Definition at line 665 of file [MemberModel.py](#).

```

00665     def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy_mod, geo_transf_ID: int, mat_ID_i = -1,
00666                   mat_ID_j = -1, ele_ID = -1):
00667         """
00668         Constructor of the class.
00669
00670         @param iNode_ID (int): ID of the first end node.
00671         @param jNode_ID (int): ID of the second end node.
00672         @param A (float): Area of the member.
00673         @param E (float): Young modulus.
00674         @param Iy_mod (float): Second moment of inertia (strong axis).
00675         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00676         documentation).
00677         @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if
00678         present). Defaults to -1.
00679         @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if
00680         present). Defaults to -1.
00681         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
00682         IDConvention to define it.
00683
00684         @exception NegativeValue: ID needs to be a positive integer.
00685         @exception NegativeValue: ID needs to be a positive integer.
00686         @exception NegativeValue: A needs to be positive.
00687         @exception NegativeValue: E needs to be positive.
00688         @exception NegativeValue: Iy_mod needs to be positive.
00689         @exception NegativeValue: ID needs to be a positive integer.
00690         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00691         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00692         @exception NameError: at least one spring needs to be defined.
00693         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00694         """
00695         # Check
00696         if iNode_ID < 1: raise NegativeValue()
00697         if jNode_ID < 1: raise NegativeValue()
00698         if A < 0: raise NegativeValue()
00699         if E < 0: raise NegativeValue()
00700         if Iy_mod < 0: raise NegativeValue()

```



```

00696         if geo_transf_ID < 1: raise NegativeValue()
00697         if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
00698         if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
00699         if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID =
{}").format(IDConvention(iNode_ID, jNode_ID))
00700         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00701
00702         # Arguments
00703         self.iNode_ID = iNode_ID
00704         self.jNode_ID = jNode_ID
00705         self.A = A
00706         self.E = E
00707         self.Iy_mod = Iy_mod
00708         self.geo_transf_ID = geo_transf_ID
00709         self.mat_ID_i = mat_ID_i
00710         self.mat_ID_j = mat_ID_j
00711
00712         # Initialized the parameters that are dependent from others
00713         self.section_name_tag = "None"
00714         self.Initialized = False
00715         self.ReInit(ele_ID)
00716
00717

```

## 7.51.3 Member Function Documentation

### 7.51.3.1 CreateMember()

```

def CreateMember (
    self )

```

Method that initialises the member by calling the OpenSeesPy commands through various functions.

Definition at line 799 of file [MemberModel.py](#).

```

00799     def CreateMember(self):
00800         """
00801         Method that initialises the member by calling the OpenSeesPy commands through various
00802         functions.
00803         """
00804         self.element_array = [[self.element_ID, self.iNode_ID_spring, self.jNode_ID_spring]]
00805         if self.mat_ID_i != -1:
00806             # Define zero length element i
00807             node(self.iNode_ID_spring, *nodeCoord(self.iNode_ID))
00808             self.iSpring_ID = IDConvention(self.iNode_ID, self.iNode_ID_spring)
00809             RotationalSpring(self.iSpring_ID, self.iNode_ID, self.iNode_ID_spring, self.mat_ID_i)
00810             self.element_array.append([self.iSpring_ID, self.iNode_ID, self.iNode_ID_spring])
00811         if self.mat_ID_j != -1:
00812             # Define zero length element j
00813             node(self.jNode_ID_spring, *nodeCoord(self.jNode_ID))
00814             self.jSpring_ID = IDConvention(self.jNode_ID, self.jNode_ID_spring)
00815             RotationalSpring(self.jSpring_ID, self.jNode_ID, self.jNode_ID_spring, self.mat_ID_j)
00816             self.element_array.append([self.jSpring_ID, self.jNode_ID, self.jNode_ID_spring])
00817
00818         # Define element
00819         element("elasticBeamColumn", self.element_ID, self.iNode_ID_spring, self.jNode_ID_spring,
00820             self.A, self.E, self.Iy_mod, self.geo_transf_ID)
00821
00822         # Update class
00823         self.Initialized = True
00824         self.UpdateStoredData()

```

### 7.51.3.2 Record()

```
def Record (
    self,
    str spring_or_element,
    str name_txt,
    str data_dir,
    force_rec = True,
    def_rec = True,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 825 of file [MemberModel.py](#).

```
00825     def Record(self, spring_or_element: str, name_txt: str, data_dir: str, force_rec=True,
00826               def_rec=True, time_rec=True):
00827         """
00828         Implementation of the homonym abstract method.
00829         See parent class MemberModel for detailed information.
00830         """
00831         if spring_or_element == "element":
00832             super().Record(self.element_ID, name_txt, data_dir, force_rec=force_rec, def_rec=def_rec,
00833                           time_rec=time_rec)
00834         elif spring_or_element == "spring_i":
00835             if self.mat_ID_i == -1:
00836                 print("Spring i recorded not present in element ID = {}".format(self.element_ID))
00837             else:
00838                 super().Record(self.iSpring_ID, name_txt, data_dir, force_rec=force_rec,
00839                               def_rec=def_rec, time_rec=time_rec)
00840         elif spring_or_element == "spring_j":
00841             if self.mat_ID_j == -1:
00842                 print("Spring j recorded not present in element ID = {}".format(self.element_ID))
00843             else:
00844                 super().Record(self.jSpring_ID, name_txt, data_dir, force_rec=force_rec,
00845                               def_rec=def_rec, time_rec=time_rec)
00846         else:
00847             print("No recording option with: '{}' with element ID: {}".format(spring_or_element,
00848                                     self.element_ID))
00849         00845
```

### 7.51.3.3 RecordNodeDef()

```
def RecordNodeDef (
    self,
    str name_txt,
    str data_dir,
    time_rec = True )
```

Implementation of the homonym abstract method.

See parent class [MemberModel](#) for detailed information.

Reimplemented from [MemberModel](#).

Definition at line 846 of file [MemberModel.py](#).

```
00846     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00847         """
00848         Implementation of the homonym abstract method.
00849         See parent class MemberModel for detailed information.
00850         """
00851         super().RecordNodeDef(self.iNode_ID, self.jNode_ID, name_txt, data_dir, time_rec=time_rec)
00852         00853
```

### 7.51.3.4 ReInit()

```
def ReInit (
    self,
    ele_ID = -1 )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use <code>IDConvention</code> to define it.
---------------	---

Definition at line 718 of file `MemberModel.py`.

```
00718     def ReInit(self, ele_ID = -1):
00719         """
00720         Implementation of the homonym abstract method.
00721         See parent class DataManagement for detailed information.
00722
00723         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
00724         IDConvention to define it.
00725         """
00726         # Members
00727         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
00728         # orientation:
00729         self.ele_orientation = NodesOrientation(self.iNode_ID, self.jNode_ID)
00730         if self.ele_orientation == "zero_length": raise ZeroLength(IDConvention(self.iNode_ID,
self.jNode_ID))
00731
00732         if self.mat_ID_i != -1:
00733             self.iNode_ID_spring = OffsetNodeIDConvention(self.iNode_ID, self.ele_orientation, "i")
00734         else:
00735             self.iNode_ID_spring = self.iNode_ID
00736
00737         if self.mat_ID_j != -1:
00738             self.jNode_ID_spring = OffsetNodeIDConvention(self.jNode_ID, self.ele_orientation, "j")
00739         else:
00740             self.jNode_ID_spring = self.jNode_ID
00741
00742         # element ID
00743         self.element_ID = IDConvention(self.iNode_ID_spring, self.jNode_ID_spring) if ele_ID == -1
00744         else ele_ID
00745
00746         # Data storage for loading/saving
00747         self.UpdateStoredData()
```

### 7.51.3.5 ShowInfo()

```
def ShowInfo (
    self,
    plot = False,
    block = False )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program Generated by DocPyGen
	Generated by DocPyGen

Definition at line 771 of file [MemberModel.py](#).

```

00771     def ShowInfo(self, plot = False, block = False):
00772         """
00773         Implementation of the homonym abstract method.
00774         See parent class DataManagement for detailed information.
00775
00776         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
00777         False.
00778         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
00779         of the program everytime that a plot should pop up). Defaults to False.
00780
00781         """
00782         print("")
00783         print("Requested info for SpringBasedElement member model, ID = {}".format(self.element_ID))
00784         print("Section associated {} ".format(self.section_name_tag))
00785         print("Material model of the spring i, ID = {}".format(self.mat_ID_i))
00786         print("Material model of the spring j, ID = {}".format(self.mat_ID_j))
00787         print("Area A = {} mm2".format(self.A/mm2_unit))
00788         print("Young modulus E = {} GPa".format(self.E/GPa_unit))
00789         print("n modified moment of inertia Iy_mod = {} mm4".format(self.Iy_mod/mm4_unit))
00790         print("Geometric transformation = {}".format(self.geo_transf_ID))
00791         print("")
00792         if plot:
00793             if self.Initialized:
00794                 plot_member(self.element_array, "SpringBased Element, ID =
00795                 {}".format(self.element_ID))
00796                 if block:
00797                     plt.show()
00798             else:
00799                 print("The SpringBasedElement is not initialized (node and elements not created), ID =
00800                 {}".format(self.element_ID))
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999

```

### 7.51.3.6 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 749 of file [MemberModel.py](#).

```

00749     def UpdateStoredData(self):
00750         """
00751         Implementation of the homonym abstract method.
00752         See parent class DataManagement for detailed information.
00753         """
00754         self.data = [{"INFO_TYPE", "SpringBasedElement"}, # Tag for differentiating different data
00755         ["element_ID", self.element_ID],
00756         ["section_name_tag", self.section_name_tag],
00757         ["A", self.A],
00758         ["E", self.E],
00759         ["Iy_mod", self.Iy_mod],
00760         ["iNode_ID", self.iNode_ID],
00761         ["iNode_ID_spring", self.iNode_ID_spring],
00762         ["mat_ID_i", self.mat_ID_i],
00763         ["jNode_ID", self.jNode_ID],
00764         ["jNode_ID_spring", self.jNode_ID_spring],
00765         ["mat_ID_j", self.mat_ID_j],
00766         ["ele_orientation", self.ele_orientation],
00767         ["tranf_ID", self.geo_transf_ID],
00768         ["Initialized", self.Initialized]]
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999

```

## 7.51.4 Member Data Documentation

#### 7.51.4.1 A

A

Definition at line 705 of file [MemberModel.py](#).

#### 7.51.4.2 data

data

Definition at line 754 of file [MemberModel.py](#).

#### 7.51.4.3 E

E

Definition at line 706 of file [MemberModel.py](#).

#### 7.51.4.4 ele\_orientation

ele\_orientation

Definition at line 728 of file [MemberModel.py](#).

#### 7.51.4.5 element\_array

element\_array

Definition at line 803 of file [MemberModel.py](#).

#### 7.51.4.6 element\_ID

element\_ID

Definition at line 742 of file [MemberModel.py](#).

#### 7.51.4.7 geo\_transf\_ID

geo\_transf\_ID

Definition at line 708 of file [MemberModel.py](#).

#### 7.51.4.8 Initialized

Initialized

Definition at line 714 of file [MemberModel.py](#).

#### 7.51.4.9 iNode\_ID

iNode\_ID

Definition at line 703 of file [MemberModel.py](#).

#### 7.51.4.10 iNode\_ID\_spring

iNode\_ID\_spring

Definition at line 732 of file [MemberModel.py](#).

#### 7.51.4.11 iSpring\_ID

iSpring\_ID

Definition at line 807 of file [MemberModel.py](#).

#### 7.51.4.12 Iy\_mod

Iy\_mod

Definition at line 707 of file [MemberModel.py](#).

#### 7.51.4.13 jNode\_ID

jNode\_ID

Definition at line 704 of file [MemberModel.py](#).

#### 7.51.4.14 jNode\_ID\_spring

jNode\_ID\_spring

Definition at line 737 of file [MemberModel.py](#).

#### 7.51.4.15 jSpring\_ID

jSpring\_ID

Definition at line 813 of file [MemberModel.py](#).

#### 7.51.4.16 mat\_ID\_i

mat\_ID\_i

Definition at line 709 of file [MemberModel.py](#).

#### 7.51.4.17 mat\_ID\_j

mat\_ID\_j

Definition at line 710 of file [MemberModel.py](#).

#### 7.51.4.18 section\_name\_tag

section\_name\_tag

Definition at line 713 of file [MemberModel.py](#).

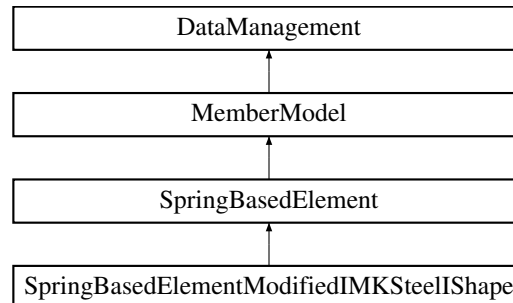
The documentation for this class was generated from the following file:

- /media/carmine/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.52 SpringBasedElementModifiedIMKSteelShape Class Reference

Class that is the children of [SpringBasedElement](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for SpringBasedElementModifiedIMKSteelShape:



### Public Member Functions

- def `__init__` (self, int `iNode_ID`, int `jNode_ID`, SteelShape `section`, int `geo_transf_ID`, new\_mat\_ID\_i=-1, new\_mat\_ID\_j=-1, N\_G=0, L\_0=-1, L\_b=-1, ele\_ID=-1)

*Constructor of the class.*

### Public Attributes

- `section`
- `section_name_tag`

#### 7.52.1 Detailed Description

Class that is the children of [SpringBasedElement](#) and combine the class SteelShape (section) to retrieve the information needed.

If there are two springs and the inflection point not in the middle, use two spring elements, connect them rigidly in the inflection point with one spring each in the extremes. `L_b` is assumed the same for top and bottom springs.

#### Parameters

<a href="#">SpringBasedElement</a>	Parent class.
------------------------------------	---------------

Definition at line 891 of file [MemberModel.py](#).

#### 7.52.2 Constructor & Destructor Documentation



7.52.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    SteelIShape section,
    int geo_transf_ID,
    new_mat_ID_i = -1,
    new_mat_ID_j = -1,
    N_G = 0,
    L_0 = -1,
    L_b = -1,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>section</i>	(SteelIShape): SteelIShape section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>new_mat_ID_i</i>	(int, optional): New ID for the definition of the material model for the spring in the node i. If -1 is passed, the class generate no material model and no spring. If 0 is passed, no i spring. Defaults to -1.
<i>new_mat_ID_j</i>	(int, optional): New ID for the definition of the material model for the spring in the node j. If -1 is passed, the class generate no material model and no spring. If 0 is passed, no j spring. Defaults to -1.
<i>N_G</i>	(float, optional): Axial load. Defaults to 0.
<i>L_0</i>	(float, optional): Distance from the maximal moment to zero. Defaults to -1, e.g. computed in <code>init()</code> .
<i>L_b</i>	(float, optional): Maximal unbraced lateral buckling length. Defaults to -1, e.g. computed in <code>init()</code> .
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NameError</i>	at least one spring needs to be defined.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>ZeroLength</i>	The two nodes are superimposed.

Reimplemented from [SpringBasedElement](#).

Definition at line 899 of file [MemberModel.py](#).

```
00900     N_G = 0, L_0 = -1, L_b = -1, ele_ID = -1):
00901     """
00902     Constructor of the class.
00903
00904     @param iNode_ID (int): ID of the first end node.
00905     @param jNode_ID (int): ID of the second end node.
00906     @param section (SteelIShape): SteelIShape section object.
00907     @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
    documentation).
```

```

00908         @param new_mat_ID_i (int, optional): New ID for the definition of the material model for the
spring in the node i.
00909         If -1 is passed, the class generate no material model and no spring. If 0 is passed, no i
spring. Defaults to -1.
00910         @param new_mat_ID_j (int, optional): New ID for the definition of the material model for the
spring in the node j.
00911         If -1 is passed, the class generate no material model and no spring. If 0 is passed, no j
spring. Defaults to -1.
00912         @param N_G (float, optional): Axial load. Defaults to 0.
00913         @param L_0 (float, optional): Distance from the maximal moment to zero. Defaults to -1, e.g.
computed in __init__().
00914         @param L_b (float, optional): Maximal unbraced lateral buckling length. Defaults to -1, e.g.
computed in __init__().
00915         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00916
00917         @exception NegativeValue: ID needs to be a positive integer.
00918         @exception NegativeValue: ID needs to be a positive integer.
00919         @exception NameError: at least one spring needs to be defined.
00920         @exception NegativeValue: ID needs to be a positive integer.
00921         @exception ZeroLength: The two nodes are superimposed.
00922         """
00923         self.section = deepcopy(section)
00924         if new_mat_ID_i != -1 and new_mat_ID_i < 0: raise NegativeValue()
00925         if new_mat_ID_j != -1 and new_mat_ID_j < 0: raise NegativeValue()
00926         if new_mat_ID_i == 0 and new_mat_ID_j == 0: raise NameError("No springs imposed for element ID
= {}). Use ElasticElement instead".format(IDConvention(iNode_ID, jNode_ID)))
00927         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00928
00929         if L_0 == -1:
00930             if new_mat_ID_i != 0 and new_mat_ID_j != 0:
00931                 L_0 = section.L/2
00932             else:
00933                 L_0 = section.L
00934         L_b = L_0 if L_b == -1 else L_b
00935
00936         # auto assign ID for material of springs
00937         ele_orientation = NodesOrientation(iNode_ID, jNode_ID)
00938         if ele_orientation == "zero_length": raise ZeroLength(IDConvention(iNode_ID, jNode_ID))
00939         if new_mat_ID_i != 0 and new_mat_ID_i == -1:
00940             new_mat_ID_i = OffsetNodeIDConvention(iNode_ID, ele_orientation, "i")
00941         if new_mat_ID_j != 0 and new_mat_ID_j == -1:
00942             new_mat_ID_j = OffsetNodeIDConvention(jNode_ID, ele_orientation, "j")
00943
00944         if new_mat_ID_i != 0:
00945             # Create mat i
00946             iSpring = ModifiedIMKSteelIShape(new_mat_ID_i, section, N_G, L_0 = L_0, L_b = L_b)
00947             iSpring.Bilin()
00948
00949         if new_mat_ID_j != 0:
00950             # Create mat j
00951             jSpring = ModifiedIMKSteelIShape(new_mat_ID_j, section, N_G, L_0 = L_0, L_b = L_b)
00952             jSpring.Bilin()
00953
00954         new_mat_ID_i = -1 if new_mat_ID_i == 0 else new_mat_ID_i
00955         new_mat_ID_j = -1 if new_mat_ID_j == 0 else new_mat_ID_j
00956
00957         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID,
mat_ID_i=new_mat_ID_i, mat_ID_j=new_mat_ID_j, ele_ID=ele_ID)
00958         self.section_name_tag = section.name_tag
00959         self.UpdateStoredData()
00960         # Check length
00961         self._CheckL()
00962
00963

```

## 7.52.3 Member Data Documentation

### 7.52.3.1 section

section

Definition at line 923 of file [MemberModel.py](#).

### 7.52.3.2 section\_name\_tag

section\_name\_tag

Definition at line 958 of file [MemberModel.py](#).

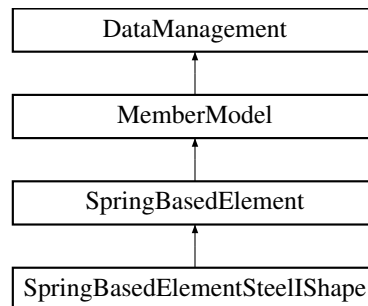
The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.53 SpringBasedElementSteelShape Class Reference

Class that is the children of [SpringBasedElement](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for SpringBasedElementSteelShape:



### Public Member Functions

- def `__init__` (self, int `iNode_ID`, int `jNode_ID`, SteelShape `section`, int `geo_transf_ID`, `mat_ID_j`=-1, `mat_ID_j`=-1, `ele_ID`=-1)

*Constructor of the class.*

### Public Attributes

- `section`
- `section_name_tag`

### 7.53.1 Detailed Description

Class that is the children of [SpringBasedElement](#) and combine the class SteelShape (section) to retrieve the information needed.

`L_b` is assumed the same for top and bottom springs.

## Parameters

<a href="#">SpringBasedElement</a>	Parent class.
------------------------------------	---------------

Definition at line 854 of file [MemberModel.py](#).

## 7.53.2 Constructor & Destructor Documentation

### 7.53.2.1 `__init__()`

```
def __init__ (
    self,
    int iNode_ID,
    int jNode_ID,
    SteelIShape section,
    int geo_transf_ID,
    mat_ID_i = -1,
    mat_ID_j = -1,
    ele_ID = -1 )
```

Constructor of the class.

## Parameters

<i>iNode_ID</i>	(int): ID of the first end node.
<i>jNode_ID</i>	(int): ID of the second end node.
<i>section</i>	(SteelIShape): SteelIShape section object.
<i>geo_transf_ID</i>	(int): A geometric transformation (for more information, see OpenSeesPy documentation).
<i>mat_ID_i</i>	(int, optional): ID of the material model for the spring in the node i (if present). Defaults to -1.
<i>mat_ID_j</i>	(int, optional): ID of the material model for the spring in the node j (if present). Defaults to -1.
<i>ele_ID</i>	(int, optional): Optional ID of the element. Defaults to -1, e.g. use IDConvention to define it.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NameError</i>	at least one spring needs to be defined.
<i>NegativeValue</i>	ID needs to be a positive integer.

Reimplemented from [SpringBasedElement](#).

Definition at line 861 of file [MemberModel.py](#).

```
00861     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int,
00862                 mat_ID_i=-1, mat_ID_j=-1, ele_ID = -1):
00863         """
00864         Constructor of the class.
00865         @param iNode_ID (int): ID of the first end node.
00866         @param jNode_ID (int): ID of the second end node.
```

```

00867         @param section (SteelIShape): SteelIShape section object.
00868         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00869         @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if
present). Defaults to -1.
00870         @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if
present). Defaults to -1.
00871         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00872
00873         @exception NegativeValue: ID needs to be a positive integer.
00874         @exception NegativeValue: ID needs to be a positive integer.
00875         @exception NameError: at least one spring needs to be defined.
00876         @exception NegativeValue: ID needs to be a positive integer.
00877         """
00878         self.section = deepcopy(section)
00879         if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
00880         if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
00881         if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID =
{}".format(IDConvention(iNode_ID, jNode_ID)))
00882         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00883
00884         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID,
mat_ID_i=mat_ID_i, mat_ID_j=mat_ID_j, ele_ID=ele_ID)
00885         self.section_name_tag = section.name_tag
00886         self.UpdateStoredData()
00887         # Check length
00888         self._CheckL()
00889
00890

```

### 7.53.3 Member Data Documentation

#### 7.53.3.1 section

section

Definition at line 878 of file [MemberModel.py](#).

#### 7.53.3.2 section\_name\_tag

section\_name\_tag

Definition at line 885 of file [MemberModel.py](#).

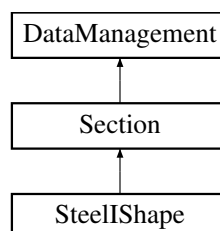
The documentation for this class was generated from the following file:

- /media/carminc/DATA/Programmi/OpenSeesPyAssistant/[MemberModel.py](#)

## 7.54 SteelShape Class Reference

Class that stores functions, geometric and mechanical properties of a steel double symmetric I-shape profile.

Inheritance diagram for SteelIShape:



## Public Member Functions

- def `__init__` (self, str `Type`, d, bf, tf, tw, L, r, E, Fy, Fy\_web=-1, name\_tag="Not Defined")  
*The constructor of the class.*
- def `Compute_iy` (self)  
*Compute the gyration radius with respect to the strong axis.*
- def `Compute_iz` (self)  
*Compute the gyration radius with respect to the weak axis.*
- def `ComputeA` (self)  
*Compute the area of a double symmetric I-profile section with fillets.*
- def `Computely` (self)  
*Compute the moment of inertia of a double symmetric I-profile section, with respect to its strong axis with fillets.*
- def `Computelz` (self)  
*Compute the moment of inertia of a double symmetric I-profile section, with respect to its weak axis with fillets.*
- def `ComputeWply` (self)  
*Compute the plastic modulus of a double symmetric I-profile section, with respect to its strong axis with fillets.*
- def `ComputeWplz` (self)  
*Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak axis with fillets.*
- def `Relnit` (self)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- `A`
- `bf`
- `d`
- `data`
- `E`
- `Fy`
- `Fy_web`
- `h_1`
- `ly`
- `iy`
- `ly_mod`
- `lz`
- `iz`
- `L`
- `My`
- `name_tag`
- `Npl`
- `r`
- `tf`
- `tw`
- `Type`
- `Wply`
- `Wplz`

## Static Public Attributes

- float `n` = 10.0

### 7.54.1 Detailed Description

Class that stores functions, geometric and mechanical properties of a steel double symmetric I-shape profile.

The parameter 'n' is used as global throughout the [SteelShape](#) sections to optimise the program (given the fact that is constant everytime).

#### Parameters

<a href="#">Section</a>	Parent abstract class.
-------------------------	------------------------

Definition at line 22 of file [Section.py](#).

### 7.54.2 Constructor & Destructor Documentation

#### 7.54.2.1 `__init__()`

```
def __init__ (
    self,
    str Type,
    d,
    bf,
    tf,
    tw,
    L,
    r,
    E,
    Fy,
    Fy_web = -1,
    name_tag = "Not Defined" )
```

The constructor of the class.

#### Parameters

<i>Type</i>	(str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
<i>d</i>	(float): Depth of the section.
<i>bf</i>	(float): Flange's width of the section
<i>tf</i>	(float): Flange's thickness of the section
<i>tw</i>	(float): Web's thickness of the section
<i>L</i>	(float): Effective length of the element associated with this section. If the panel zone is present, exclude its dimension.
<i>r</i>	(float): Radius of the weld fillets of the section.
<i>E</i>	(float): Young modulus of the section.
<i>Fy</i>	(float): Yield strength of the flange of the section. Used as the yield strength of the entire section.
<i>Fy_web</i>	(float, optional): Yield strength of the web of the section. Used for panel zone associated to this section. Defaults to -1, e.g. computed in <code>init()</code> as equal to Fy.
<i>name_tag</i>	(str, optional): Name TAG of the section. Defaults to "Not Defined".

## Exceptions

<i>WrongArgument</i>	Type needs to be 'Col' or 'Beam'.
<i>NegativeValue</i>	d needs to be positive.
<i>NegativeValue</i>	bf needs to be positive.
<i>NegativeValue</i>	tf needs to be positive.
<i>NegativeValue</i>	tw needs to be positive.
<i>NegativeValue</i>	L needs to be positive.
<i>NegativeValue</i>	r needs to be positive.
<i>NegativeValue</i>	E needs to be positive.
<i>NegativeValue</i>	Fy needs to be positive.
<i>NegativeValue</i>	Fy_web needs to be positive if different from -1.
<i>InconsistentGeometry</i>	tw should be smaller than bf.
<i>InconsistentGeometry</i>	tf needs to be smaller than half of d
<i>InconsistentGeometry</i>	r should be less than half bf and d

Definition at line 32 of file [Section.py](#).

```

00032     def __init__(self, Type: str, d, bf, tf, tw, L, r, E, Fy, Fy_web = -1, name_tag = "Not Defined"):
00033         """
00034         The constructor of the class.
00035
00036         @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
00037         @param d (float): Depth of the section.
00038         @param bf (float): Flange's width of the section
00039         @param tf (float): Flange's thickness of the section
00040         @param tw (float): Web's thickness of the section
00041         @param L (float): Effective length of the element associated with this section.
00042             If the panel zone is present, exclude its dimension.
00043         @param r (float): Radius of the weld fillets of the section.
00044         @param E (float): Young modulus of the section.
00045         @param Fy (float): Yield strength of the flange of the section. Used as the yield strength of
the entire section.
00046         @param Fy_web (float, optional): Yield strength of the web of the section. Used for panel zone
associated to this section.
00047             Defaults to -1, e.g. computed in __init__() as equal to Fy.
00048         @param name_tag (str, optional): Name TAG of the section. Defaults to "Not Defined".
00049
00050         @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
00051         @exception NegativeValue: d needs to be positive.
00052         @exception NegativeValue: bf needs to be positive.
00053         @exception NegativeValue: tf needs to be positive.
00054         @exception NegativeValue: tw needs to be positive.
00055         @exception NegativeValue: L needs to be positive.
00056         @exception NegativeValue: r needs to be positive.
00057         @exception NegativeValue: E needs to be positive.
00058         @exception NegativeValue: Fy needs to be positive.
00059         @exception NegativeValue: Fy_web needs to be positive if different from -1.
00060         @exception InconsistentGeometry: tw should be smaller than bf.
00061         @exception InconsistentGeometry: tf needs to be smaller than half of d
00062         @exception InconsistentGeometry: r should be less than half bf and d
00063         """
00064         # Check
00065         if Type != "Beam" and Type != "Col": raise WrongArgument()
00066         if d < 0: raise NegativeValue()
00067         if bf < 0: raise NegativeValue()
00068         if tf < 0: raise NegativeValue()
00069         if tw < 0: raise NegativeValue()
00070         if L < 0: raise NegativeValue()
00071         if r < 0: raise NegativeValue()
00072         if E < 0: raise NegativeValue()
00073         if Fy < 0: raise NegativeValue()
00074         if Fy_web != -1 and Fy_web < 0: raise NegativeValue()
00075         if tw > bf: raise InconsistentGeometry()
00076         if tf > d/2: raise InconsistentGeometry()
00077         if r > bf/2 or r > d/2: raise InconsistentGeometry()
00078
00079         # Arguments
00080         self.Type = Type
00081         self.d = d
00082         self.bf = bf
00083         self.tf = tf
00084         self.tw = tw
00085         self.L = L

```



```

00086         self.r = r
00087         self.E = E
00088         self.Fy = Fy
00089         self.Fy_web = Fy if Fy_web == -1 else Fy_web
00090         self.name_tag = name_tag
00091
00092         # Initialized the parameters that are dependent from others
00093         self.ReInit()
00094

```

### 7.54.3 Member Function Documentation

#### 7.54.3.1 Compute\_iy()

```

def Compute_iy (
    self )

```

Compute the gyration radius with respect to the strong axis.

##### Returns

float: The gyration radius with respect to the strong axis.

Definition at line 241 of file [Section.py](#).

```

00241     def Compute_iy(self):
00242         """
00243         Compute the gyration radius with respect to the strong axis.
00244
00245         @returns float: The gyration radius with respect to the strong axis.
00246         """
00247         # Iy :    The second moment of inertia with respect to thte strong axis
00248         # A :    The area
00249
00250         return math.sqrt(self.Iy/self.A)
00251

```

#### 7.54.3.2 Compute\_iz()

```

def Compute_iz (
    self )

```

Compute the gyration radius with respect to the weak axis.

##### Returns

float: The gyration radius with respect to the weak axis.

Definition at line 252 of file [Section.py](#).

```

00252     def Compute_iz(self):
00253         """
00254         Compute the gyration radius with respect to the weak axis.
00255
00256         @returns float: The gyration radius with respect to the weak axis.
00257         """
00258         # Iz :    The second moment of inertia with respect to thte weak axis
00259         # A :    The area
00260
00261         return math.sqrt(self.Iz/self.A)
00262
00263

```

### 7.54.3.3 ComputeA()

```
def ComputeA (
    self )
```

Compute the area of a double symmetric I-profile section with fillets.

#### Returns

float: Area of the I shape section (with fillets included)

Definition at line 170 of file [Section.py](#).

```
00170     def ComputeA(self):
00171         """
00172         Compute the area of a double symmetric I-profile section with fillets.
00173
00174         @returns float: Area of the I shape section (with fillets included)
00175         """
00176         # d :      The depth
00177         # bf :      The flange's width
00178         # tf :      The flange's thickness
00179         # tw :      The web's thickness
00180         # r :      The weld fillet radius
00181
00182         # without fillets bf*tf*2 + tw*(d-2*tf)
00183         return 2.0*self.bf*self.tf+self.tw*(self.d-2.0*self.tf)+0.8584*self.r**2
00184
```

### 7.54.3.4 Computely()

```
def Computely (
    self )
```

Compute the moment of inertia of a double symmetric I-profile section, with respect to its strong axis with fillets.

#### Returns

float: The moment of inertia with respect to the strong axis.

Definition at line 185 of file [Section.py](#).

```
00185     def Computely(self):
00186         """
00187         Compute the moment of inertia of a double symmetric I-profile section, with respect to its
00188         strong axis with fillets.
00189
00189         @returns float: The moment of inertia with respect to the strong axis.
00190         """
00191         # d :      The depth
00192         # bf :      The flange's width
00193         # tf :      The flange's thickness
00194         # tw :      The web's thickness
00195         # r :      The weld fillet radius
00196
00197         # without fillets: bf*tf/2*(d-tf)**2 + bf*tf**3/6 + (d-tf*2)**3*tf/12
00198         return
00199         (self.bf*self.d**3.0-(self.bf-self.tw)*(self.d-2.0*self.tf)**3)/12.0+0.8584*self.r**2*(0.5*self.d-self.tf-0.4467*self.r)
```

### 7.54.3.5 ComputeIz()

```
def ComputeIz (
    self )
```

Compute the moment of inertia of a double symmetric I-profile section, with respect to its weak axis with fillets.

#### Returns

float: The moment of inertia with respect to the weak axis.

Definition at line 200 of file [Section.py](#).

```
00200     def ComputeIz(self):
00201         """
00202         Compute the moment of inertia of a double symmetric I-profile section, with respect to its
00203         weak axis with fillets.
00204
00205         @returns float: The moment of inertia with respect to the weak axis.
00206         """
00207         # d :      The depth
00208         # bf :     The flange's width
00209         # tf :     The flange's thickness
00210         # tw :     The web's thickness
00211         # r :      The weld fillet radius
00212
00213         return
00214         (self.tf*self.bf**3)/6.0+((self.d-2.0*self.tf)*self.tw**3)/12.0+0.8584*self.r**2*(0.5*self.tw+0.2234*self.r)**2
```

### 7.54.3.6 ComputeWply()

```
def ComputeWply (
    self )
```

Compute the plastic modulus of a double symmetric I-profile section, with respect to its strong axis with fillets.

#### Returns

float: The plastic modulus with respect to the strong axis.

Definition at line 214 of file [Section.py](#).

```
00214     def ComputeWply(self):
00215         """
00216         Compute the plastic modulus of a double symmetric I-profile section, with respect to its
00217         strong axis with fillets.
00218
00219         @returns float: The plastic modulus with respect to the strong axis.
00220         """
00221         # d :      The depth
00222         # bf :     The flange's width
00223         # tf :     The flange's thickness
00224         # tw :     The web's thickness
00225         # r :      The weld fillet radius
00226
00227         return
00228         self.bf*self.tf*(self.d-self.tf)+(self.d-2.0*self.tf)**2.0*(self.tw/4.0)+0.4292*self.r**2*(self.d-2.0*self.tf-0.4467*self.r)
```

### 7.54.3.7 ComputeWplz()

```
def ComputeWplz (
    self )
```

Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak axis with fillets.

#### Returns

float: The plastic modulus with respect to the weak axis.

Definition at line 228 of file [Section.py](#).

```
00228     def ComputeWplz(self):
00229         """
00230         Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak
axis with fillets.
00231
00232         @returns float: The plastic modulus with respect to the weak axis.
00233         """
00234         # d :      The depth
00235         # bf :     The flange's width
00236         # tf :     The flange's thickness
00237         # tw :     The web's thickness
00238         # r :      The weld fillet radius
00239         return
(self.tf*self.bf**2)/2+(self.d-2.0*self.tf)*(self.tw**2/4.0)+0.4292*self.r**2*(self.tw+0.4467*self.r)
00240
```

### 7.54.3.8 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 95 of file [Section.py](#).

```
00095     def ReInit(self):
00096         """
00097         Implementation of the homonym abstract method.
00098         See parent class DataManagement for detailed information.
00099         """
00100         # Member
00101         self.h_1 = self.d - 2.0*self.r - 2.0*self.tf
00102         self.A = self.ComputeA()
00103         self.Npl = self.A*self.Fy
00104         self.Iy = self.ComputeIy()
00105         self.Iz = self.ComputeIz()
00106         self.Wply = self.ComputeWply()
00107         self.Wplz = self.ComputeWplz()
00108         self.My = self.Fy*self.Wply
00109         self.Iy_mod = self.Iy*(n + 1.0)/n
00110         self.iz = self.Compute_iz()
00111         self.iy = self.Compute_iy()
00112
00113         # Data storage for loading/saving
00114         self.UpdateStoredData()
00115
```

### 7.54.3.9 ShowInfo()

```
def ShowInfo (
                self )
```

### Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 145 of file Section.py.

```

00145 def ShowInfo(self):
00146     """
00147     Implementation of the homonym abstract method.
00148     See parent class DataManagement for detailed information.
00149     """
00150     print("")
00151     print("Requested info for steel I shape section of type = {} and name tag = {}".format(self.Type, self.name_tag))
00152     print("d = {} mm".format(self.d/mm_unit))
00153     print("Fy = {} MPa".format(self.Fy/MPa_unit))
00154     print("Fy web = {} MPa".format(self.Fy_web/MPa_unit))
00155     print("E = {} GPa".format(self.E/GPa_unit))
00156     print("h_1 = {} mm".format(self.h_1/mm_unit))
00157     print("A = {} mm2".format(self.A/mm2_unit))
00158     print("Iy = {} mm4".format(self.Iy/mm4_unit))
00159     print("Iz = {} mm4".format(self.Iz/mm4_unit))
00160     print("Wply = {} mm3".format(self.Wply/mm3_unit))
00161     print("Wplz = {} mm3".format(self.Wplz/mm3_unit))
00162     print("Iy_mod = {} mm4".format(self.Iy_mod/mm4_unit))
00163     print("iy = {} mm".format(self.iy/mm_unit))
00164     print("iz = {} mm".format(self.iz/mm_unit))
00165     print("My = {} kNm".format(self.My/kNm_unit))
00166     print("Npl = {} kN".format(self.Npl/kN_unit))
00167     print("")
00168
00169

```

### 7.54.3.10 UpdateStoredData()

```
def UpdateStoredData (
    self )
```

### Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 116 of file Section.py.

```

00116     def UpdateStoredData(self):
00117         """
00118         Implementation of the homonym abstract method.
00119         See parent class DataManagement for detailed information.
00120         """
00121         self.data = [{"INFO_TYPE", "SteelIShape"}, # Tag for differentiating different data
00122                     {"name_tag", self.name_tag},
00123                     {"Type", self.Type},
00124                     {"d", self.d},
00125                     {"bf", self.bf},
00126                     {"tf", self.tf},
00127                     {"tw", self.tw},
00128                     {"L", self.L},
00129                     {"r", self.r},
00130                     {"h_1", self.h_1},
00131                     {"E", self.E},
00132                     {"Fy", self.Fy},
00133                     {"Fy_web", self.Fy_web},
00134                     {"A", self.A},
00135                     {"Iy", self.Iy},
00136                     {"Iz", self.Iz},
00137                     {"Wply", self.Wply},
00138                     {"Wplz", self.Wplz},
00139                     {"Iy_mod", self.Iy_mod},
00140                     {"iy", self.iy},
00141                     {"iz", self.iz},
00142                     {"Npl", self.Npl},
00143                     {"My", self.My}]
00144

```

## 7.54.4 Member Data Documentation

### 7.54.4.1 A

A

Definition at line 102 of file [Section.py](#).

### 7.54.4.2 bf

bf

Definition at line 82 of file [Section.py](#).

### 7.54.4.3 d

d

Definition at line 81 of file [Section.py](#).

### 7.54.4.4 data

data

Definition at line 121 of file [Section.py](#).

### 7.54.4.5 E

E

Definition at line 87 of file [Section.py](#).

#### 7.54.4.6 **Fy**

Fy

Definition at line 88 of file [Section.py](#).

#### 7.54.4.7 **Fy\_web**

Fy\_web

Definition at line 89 of file [Section.py](#).

#### 7.54.4.8 **h\_1**

h\_1

Definition at line 101 of file [Section.py](#).

#### 7.54.4.9 **Iy**

Iy

Definition at line 104 of file [Section.py](#).

#### 7.54.4.10 **iy**

iy

Definition at line 111 of file [Section.py](#).

#### 7.54.4.11 **Iy\_mod**

Iy\_mod

Definition at line 109 of file [Section.py](#).

**7.54.4.12 Iz**

Iz

Definition at line 105 of file [Section.py](#).

**7.54.4.13 iz**

iz

Definition at line 110 of file [Section.py](#).

**7.54.4.14 L**

L

Definition at line 85 of file [Section.py](#).

**7.54.4.15 My**

My

Definition at line 108 of file [Section.py](#).

**7.54.4.16 n**

```
float n = 10.0 [static]
```

Definition at line 30 of file [Section.py](#).

**7.54.4.17 name\_tag**

name\_tag

Definition at line 90 of file [Section.py](#).



**7.54.4.18 Npl**

Npl

Definition at line 103 of file [Section.py](#).

**7.54.4.19 r**

r

Definition at line 86 of file [Section.py](#).

**7.54.4.20 tf**

tf

Definition at line 83 of file [Section.py](#).

**7.54.4.21 tw**

tw

Definition at line 84 of file [Section.py](#).

**7.54.4.22 Type**

Type

Definition at line 80 of file [Section.py](#).

**7.54.4.23 Wply**

Wply

Definition at line 106 of file [Section.py](#).

#### 7.54.4.24 Wplz

Wplz

Definition at line 107 of file [Section.py](#).

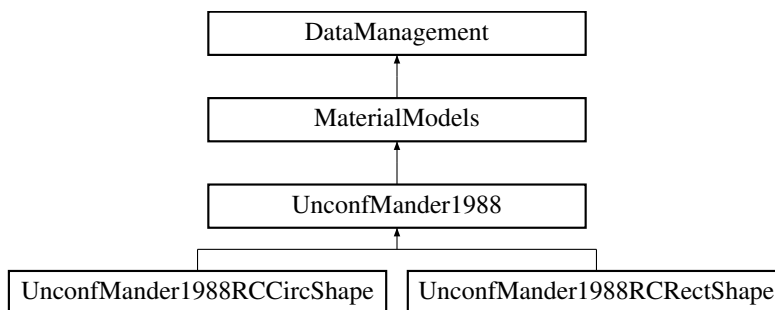
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py](#)

### 7.55 UnconfMander1988 Class Reference

Class that stores functions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

Inheritance diagram for UnconfMander1988:



#### Public Member Functions

- def `__init__` (self, int ID, fc, Ec, ec=1, ecp=1, fct=-1, et=-1, beta=0.1)  
*Constructor of the class.*
- def `CheckApplicability` (self)  
*Implementation of the homonym abstract method.*
- def `Compute_ec` (self)  
*Method that computes the compressive concrete yield strain.*
- def `Compute_ecp` (self)  
*Method that computes the compressive concrete spalling strain.*
- def `Compute_ecu` (self)  
*Method that computes the compressive concrete failure strain.*
- def `Compute_et` (self)  
*Method that computes the tensile concrete yield strain.*
- def `Compute_fct` (self)  
*Method that computes the tensile concrete yield stress.*
- def `Concrete01` (self)  
*Generate the material model Concrete01 for unconfined concrete using the computed parameters.*
- def `Concrete04` (self)  
*Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the computed parameters.*
- def `Relnit` (self, ec=1, ecp=1, fct=-1, et=-1)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, plot=False, block=False, concrete04=True)  
*Implementation of the homonym abstract method.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- [beta](#)
- [data](#)
- [Ec](#)
- [ec](#)
- [ecp](#)
- [ecu](#)
- [et](#)
- [fc](#)
- [fct](#)
- [ID](#)
- [Initialized](#)
- [section\\_name\\_tag](#)

### 7.55.1 Detailed Description

Class that stores functions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.

For more information about the empirical model for the computation of the parameters, see Mander et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 1117 of file [MaterialModels.py](#).

### 7.55.2 Constructor & Destructor Documentation

#### 7.55.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int ID,
    fc,
    Ec,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    beta = 0.1 )
```

Constructor of the class.

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>fc</i>	(float): Compressive concrete yield strength (needs to be negative).
<i>Ec</i>	(float): Young modulus.
<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>PositiveValue</i>	fc needs to be negative.
<i>NegativeValue</i>	Ec needs to be positive.
<i>PositiveValue</i>	ec needs to be negative if different from 1.
<i>PositiveValue</i>	ecp needs to be positive if different from 1.
<i>NegativeValue</i>	fct needs to be positive if different from -1.
<i>NegativeValue</i>	et needs to be positive if different from -1.

Reimplemented in [UnconfMander1988RCCircShape](#), and [UnconfMander1988RCRectShape](#).

Definition at line 1125 of file [MaterialModels.py](#).

```

01125     def __init__(self, ID: int, fc, Ec, ec = 1, ecp = 1, fct = -1, et = -1, beta = 0.1):
01126         """
01127         Constructor of the class.
01128
01129         @param ID (int): Unique material model ID.
01130         @param fc (float): Compressive concrete yield strength (needs to be negative).
01131         @param Ec (float): Young modulus.
01132         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01133         according to Karthik and Mander 2011.
01134         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01135         to Mander 1988.
01136         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01137         according to SIA 262:2012.
01138         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01139         according to SIA 262:2012.
01140         @param beta (float, optional): Loading point value defining the exponential curve parameter to
01141         define the residual stress.
01142         Defaults to 0.1 (according to OpenSeesPy documentation)
01143
01144         @exception NegativeValue: ID needs to be a positive integer.
01145         @exception PositiveValue: fc needs to be negative.
01146         @exception NegativeValue: Ec needs to be positive.
01147         @exception PositiveValue: ec needs to be negative if different from 1.
01148         @exception PositiveValue: ecp needs to be positive if different from 1.
01149         @exception NegativeValue: fct needs to be positive if different from -1.
01150         @exception NegativeValue: et needs to be positive if different from -1.
01151         """
01152         # Check
01153         if ID < 0: raise NegativeValue()
01154         if fc > 0: raise PositiveValue()
01155         if Ec < 0: raise NegativeValue()
01156         if ec != 1 and ec > 0: raise PositiveValue()
01157         if ecp != 1 and ecp > 0: raise PositiveValue()
01158         if fct != -1 and fct < 0: raise NegativeValue()
01159         if et != -1 and et < 0: raise NegativeValue()
01160
01161         # Arguments
01162         self.ID = ID
01163         self.fc = fc
01164         self.Ec = Ec
01165         self.beta = beta

```

```

01161
01162     # Initialized the parameters that are dependent from others
01163     self.section_name_tag = "None"
01164     self.Initialized = False
01165     self.ReInit(ec, ecp, fct, et)
01166

```

## 7.55.3 Member Function Documentation

### 7.55.3.1 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 1242 of file [MaterialModels.py](#).

```

01242     def CheckApplicability(self):
01243         """
01244         Implementation of the homonym abstract method.
01245         See parent class MaterialModels for detailed information.
01246         """
01247         Check = True
01248         if self.fc < -110*MPa_unit: # Deierlein 1999
01249             Check = False
01250             print("With High Strength concrete (< -110 MPa), a better material model should be used
01251             (see Abdesselam et Al. 2019)")
01252         if not Check:
01253             print("The validity of the equations is not fullfilled.")
01254             print("!!!!!! WARNING !!!!!!! Check material model of Unconfined Mander 1988, ID=",
01255             self.ID)
01256             print("")

```

### 7.55.3.2 Compute\_ec()

```

def Compute_ec (
    self )

```

Method that computes the compressive concrete yield strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 1257 of file [MaterialModels.py](#).

```

01257     def Compute_ec(self):
01258         """
01259         Method that computes the compressive concrete yield strain.
01260         For more information, see Karthik and Mander 2011.
01261
01262         @returns float: Strain
01263         """
01264         # return -0.002 # Alternative: Mander et Al. 1988
01265         return -0.0015 + self.fc/MPa_unit/70000 # Karthik Mander 2011
01266

```

### 7.55.3.3 Compute\_ecp()

```
def Compute_ecp (
    self )
```

Method that computes the compressive concrete spalling strain.

For more information, see Mander et Al. 1988.

#### Returns

float: Strain

Definition at line 1267 of file [MaterialModels.py](#).

```
01267     def Compute_ecp(self):
01268         """
01269         Method that computes the compressive concrete spalling strain.
01270         For more information, see Mander et Al. 1988.
01271
01272         @returns float: Strain
01273         """
01274         return 2.0*self.ec
01275
01276
```

### 7.55.3.4 Compute\_ecu()

```
def Compute_ecu (
    self )
```

Method that computes the compressive concrete failure strain.

For more information, see Karthik and Mander 2011.

#### Returns

float: Strain

Definition at line 1297 of file [MaterialModels.py](#).

```
01297     def Compute_ecu(self):
01298         """
01299         Method that computes the compressive concrete failure strain.
01300         For more information, see Karthik and Mander 2011.
01301
01302         @returns float: Strain
01303         """
01304         # return -0.004 # Alternative: Mander et Al. 1988
01305         return -0.012 - 0.0001 * self.fc/MPa_unit # Karthik Mander 2011
01306
```

### 7.55.3.5 Compute\_et()

```
def Compute_et (
    self )
```

Method that computes the tensile concrete yield strain.

For more information, see Mander et Al. 1988 (eq 45).

#### Returns

float: Strain.

Definition at line 1287 of file [MaterialModels.py](#).

```
01287     def Compute_et(self):
01288         """
01289         Method that computes the tensile concrete yield strain.
01290         For more information, see Mander et Al. 1988 (eq 45).
01291
01292         @returns float: Strain.
01293         """
01294         return self.fct/self.Ec
01295
01296
```

### 7.55.3.6 Compute\_fct()

```
def Compute_fct (
    self )
```

Method that computes the tensile concrete yield stress.

For more information, see SIA 262:2012.

#### Returns

float: Stress.

Definition at line 1277 of file [MaterialModels.py](#).

```
01277     def Compute_fct(self):
01278         """
01279         Method that computes the tensile concrete yield stress.
01280         For more information, see SIA 262:2012.
01281
01282         @returns float: Stress.
01283         """
01284         return 0.30 * math.pow(-self.fc/MPa_unit, 2/3) * MPa_unit
01285
01286
```

### 7.55.3.7 Concrete01()

```
def Concrete01 (
    self )
```

Generate the material model Concrete01 for unconfined concrete using the computed parameters.

See `_Concrete01` function for more information. Use this method or `Concrete04`, not both (only one material model for ID).

Definition at line 1307 of file [MaterialModels.py](#).

```
01307     def Concrete01(self):
01308         """
01309         Generate the material model Concrete01 for unconfined concrete using the computed parameters.
01310         See _Concrete01 function for more information. Use this method or Concrete04, not both (only
01311         one material model for ID).
01312         """
01312         _Concrete01(self.ID, self.ec, self.fc, self.ecu)
01313         self.Initialized = True
01314         self.UpdateStoredData()
01315
01316
```

### 7.55.3.8 Concrete04()

```
def Concrete04 (
    self )
```

Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the computed parameters.

See `_Concrete04` function for more information. Use this method or `Concrete01`, not both (only one material model for ID).

Definition at line 1317 of file [MaterialModels.py](#).

```
01317     def Concrete04(self):
01318         """
01319         Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the
01320         computed parameters.
01320         See _Concrete04 function for more information. Use this method or Concrete01, not both (only
01321         one material model for ID).
01322         """
01322         _Concrete04(self.ID, self.fc, self.ec, self.ecu, self.Ec, self.fct, self.et, self.beta)
01323         self.Initialized = True
01324         self.UpdateStoredData()
01325
01326
```

### 7.55.3.9 ReInit()

```
def ReInit (
    self,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1 )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.



## Parameters

<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.

Definition at line 1168 of file [MaterialModels.py](#).

```

01168     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
01169         """
01170         Implementation of the homonym abstract method.
01171         See parent class DataManagement for detailed information.
01172
01173         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01174         according to Karthik and Mander 2011.
01175         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01176         to Mander 1988.
01177         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01178         according to SIA 262:2012.
01179         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01180         according to SIA 262:2012.
01181         """
01182         # Check applicability
01183         self.CheckApplicability()
01184
01185         # Arguments
01186         self.ec = self.Compute_ec() if ec == 1 else ec
01187         self.ecp = self.Compute_ecp() if ecp == 1 else ecp
01188         self.fct = self.Compute_fct() if fct == -1 else fct
01189         self.et = self.Compute_et() if et == -1 else et
01190
01191         # Members
01192         self.ecu = self.Compute_ecu()
01193         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
01194         (modified)"
01195
01196         # Data storage for loading/saving
01197         self.UpdateStoredData()
01198

```

## 7.55.3.10 ShowInfo()

```

def ShowInfo (
    self,
    plot = False,
    block = False,
    concrete04 = True )

```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

## Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.
<i>concrete04</i>	(bool, optional): Option to show in the plot the concrete04 or concrete01 if False. Defaults to True.

Definition at line 1214 of file [MaterialModels.py](#).

```

01214     def ShowInfo(self, plot = False, block = False, concrete04 = True):
01215         """
01216         Implementation of the homonym abstract method.
01217         See parent class DataManagement for detailed information.
01218
01219         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
01220         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
01221         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if
False. Defaults to True.
01222         """
01223         print("")
01224         print("Requested info for Unconfined Mander 1988 material model Parameters, ID =
{}".format(self.ID))
01225         print("Section associated: {} ".format(self.section_name_tag))
01226         print('Concrete strength fc = {} MPa'.format(self.fc/MPa_unit))
01227         print('Strain at maximal strength ec = {}'.format(self.ec))
01228         print('Maximal strain ecu = {}'.format(self.ecu))
01229         print("")
01230
01231         if plot:
01232             fig, ax = plt.subplots()
01233             if concrete04:
01234                 PlotConcrete04(self.fc, self.Ec, self.ec, self.ecu, "U", ax, self.ID)
01235             else:
01236                 PlotConcrete01(self.fc, self.ec, 0, self.ecu, ax, self.ID)
01237
01238             if block:
01239                 plt.show()
01240
01241

```

### 7.55.3.11 UpdateStoredData()

```

def UpdateStoredData (
    self )

```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 1195 of file `MaterialModels.py`.

```

01195     def UpdateStoredData(self):
01196         """
01197         Implementation of the homonym abstract method.
01198         See parent class DataManagement for detailed information.
01199         """
01200         self.data = [{"INFO_TYPE", "UnconfMander1988"}, # Tag for differentiating different data
01201                     [{"ID", self.ID},
01202                      ["section_name_tag", self.section_name_tag],
01203                      ["fc", self.fc],
01204                      ["Ec", self.Ec],
01205                      ["ec", self.ec],
01206                      ["ecp", self.ecp],
01207                      ["ecu", self.ecu],
01208                      ["fct", self.fct],
01209                      ["et", self.et],
01210                      ["beta", self.beta],
01211                      ["Initialized", self.Initialized]]
01212
01213

```

## 7.55.4 Member Data Documentation

#### 7.55.4.1 beta

beta

Definition at line 1160 of file [MaterialModels.py](#).

#### 7.55.4.2 data

data

Definition at line 1200 of file [MaterialModels.py](#).

#### 7.55.4.3 Ec

Ec

Definition at line 1159 of file [MaterialModels.py](#).

#### 7.55.4.4 ec

ec

Definition at line 1182 of file [MaterialModels.py](#).

#### 7.55.4.5 ecp

ecp

Definition at line 1183 of file [MaterialModels.py](#).

#### 7.55.4.6 ecu

ecu

Definition at line 1188 of file [MaterialModels.py](#).

#### 7.55.4.7 et

et

Definition at line 1185 of file [MaterialModels.py](#).

#### 7.55.4.8 fc

fc

Definition at line 1158 of file [MaterialModels.py](#).

#### 7.55.4.9 fct

fct

Definition at line 1184 of file [MaterialModels.py](#).

#### 7.55.4.10 ID

ID

Definition at line 1157 of file [MaterialModels.py](#).

#### 7.55.4.11 Initialized

Initialized

Definition at line 1164 of file [MaterialModels.py](#).

#### 7.55.4.12 section\_name\_tag

section\_name\_tag

Definition at line 1163 of file [MaterialModels.py](#).

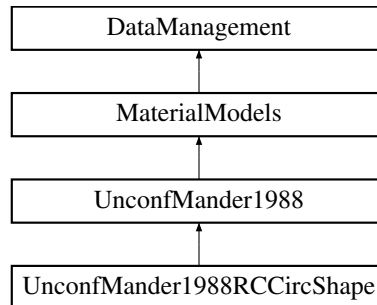
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.56 UnconfMander1988RCCircShape Class Reference

Class that is the children of [UnconfMander1988](#) and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for UnconfMander1988RCCircShape:



### Public Member Functions

- `def __init__(self, int ID, RCCircShape section, ec=1, ecp=1, fct=-1, et=-1, beta=0.1)`  
*Constructor of the class.*

### Public Attributes

- `section`
- `section_name_tag`

#### 7.56.1 Detailed Description

Class that is the children of [UnconfMander1988](#) and combine the class RCCircShape (section) to retrieve the information needed.

#### Parameters

<a href="#">UnconfMander1988</a>	Parent class.
----------------------------------	---------------

Definition at line 1354 of file [MaterialModels.py](#).

#### 7.56.2 Constructor & Destructor Documentation

### 7.56.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RCCircShape section,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCCircShape. The copy of the section passed is stored in the member variable self.section.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(RCCircShape): RCCircShape section object.
<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from [UnconfMander1988](#).

Definition at line 1360 of file [MaterialModels.py](#).

```
01360     def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
01361         """
01362         Constructor of the class. It passes the arguments into the parent class to generate the
01363         combination of the parent class
01364         and the section class RCCircShape.
01365         The copy of the section passed is stored in the member variable self.section.
01366
01367         @param ID (int): Unique material model ID.
01368         @param section (RCCircShape): RCCircShape section object.
01369         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01370         according to Karthik and Mander 2011.
01371         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01372         to Mander 1988.
01373         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01374         according to SIA 262:2012.
01375         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01376         according to SIA 262:2012.
01377         @param beta (float, optional): Loading point value defining the exponential curve parameter to
01378         define the residual stress.
01379         Defaults to 0.1 (according to OpenSeesPy documentation)
01380         """
01381         self.section = deepcopy(section)
01382         super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
01383         self.section_name_tag = section.name_tag
01384         self.UpdateStoredData()
```

## 7.56.3 Member Data Documentation

### 7.56.3.1 section

section

Definition at line 1375 of file [MaterialModels.py](#).

### 7.56.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1377 of file [MaterialModels.py](#).

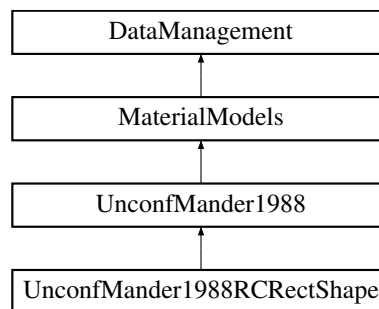
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.57 UnconfMander1988RCRectShape Class Reference

Class that is the children of [UnconfMander1988](#) and combine the class RCRectShape (section) to retrieve the information needed.

Inheritance diagram for UnconfMander1988RCRectShape:



### Public Member Functions

- `def __init__(self, int ID, RCRectShape section, ec=1, ecp=1, fct=-1, et=-1, beta=0.1)`  
*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

### 7.57.1 Detailed Description

Class that is the children of [UnconfMander1988](#) and combine the class RCRectShape (section) to retrieve the information needed.

## Parameters

<a href="#">UnconfMander1988</a>	Parent class.
----------------------------------	---------------

Definition at line 1327 of file [MaterialModels.py](#).

## 7.57.2 Constructor & Destructor Documentation

### 7.57.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int ID,
    RCRectShape section,
    ec = 1,
    ecp = 1,
    fct = -1,
    et = -1,
    beta = 0.1 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class RCRectShape. The copy of the section passed is stored in the member variable self.section.

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(RCRectShape): RCRectShape section object.
<i>ec</i>	(float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed according to Karthik and Mander 2011.
<i>ecp</i>	(float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according to Mander 1988.
<i>fct</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>et</i>	(float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed according to SIA 262:2012.
<i>beta</i>	(float, optional): Loading point value defining the exponential curve parameter to define the residual stress. Defaults to 0.1 (according to OpenSeesPy documentation)

Reimplemented from [UnconfMander1988](#).

Definition at line 1333 of file [MaterialModels.py](#).

```
01333 def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
01334     """
01335     Constructor of the class. It passes the arguments into the parent class to generate the
    combination of the parent class
01336     and the section class RCRectShape.
01337     The copy of the section passed is stored in the member variable self.section.
01338
01339     @param ID (int): Unique material model ID.
01340     @param section (RCRectShape): RCRectShape section object.
```



```

01341         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01342         according to Karthik and Mander 2011.
01343         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01344         to Mander 1988.
01345         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01346         according to SIA 262:2012.
01347         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01348         according to SIA 262:2012.
01349         @param beta (float, optional): Loading point value defining the exponential curve parameter to
01350         define the residual stress.
01351         Defaults to 0.1 (according to OpenSeesPy documentation)
01352         """
01353         self.section = deepcopy(section)
01354         super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
01355         self.section_name_tag = section.name_tag
01356         self.UpdateStoredData()

```

### 7.57.3 Member Data Documentation

#### 7.57.3.1 section

section

Definition at line 1348 of file [MaterialModels.py](#).

#### 7.57.3.2 section\_name\_tag

section\_name\_tag

Definition at line 1350 of file [MaterialModels.py](#).

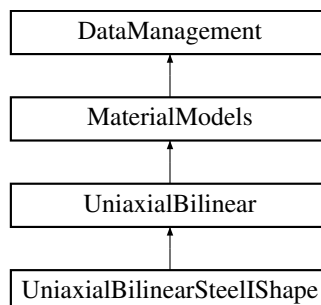
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.58 UniaxialBilinear Class Reference

Class that stores functions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

Inheritance diagram for UniaxialBilinear:



## Public Member Functions

- def `__init__` (self, int `ID`, `fy`, `Ey`, `b`=0.01)  
*Constructor of the class.*
- def `CheckApplicability` (self)  
*Implementation of the homonym abstract method.*
- def `Relnit` (self)  
*Implementation of the homonym abstract method.*
- def `ShowInfo` (self, `plot`=False, `block`=False)  
*Implementation of the homonym abstract method.*
- def `Steel01` (self)  
*Generate the material model Steel01 uniaxial bilinear material model.*
- def `UpdateStoredData` (self)  
*Implementation of the homonym abstract method.*

## Public Attributes

- `b`
- `data`
- `Ey`
- `ey`
- `fy`
- `ID`
- `Initialized`
- `section_name_tag`

### 7.58.1 Detailed Description

Class that stores functions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.

#### Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 2319 of file [MaterialModels.py](#).

### 7.58.2 Constructor & Destructor Documentation

#### 7.58.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    fy,
```

$$E_y,$$
$$b = 0.01 )$$

Constructor of the class.

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>fy</i>	(float): Yield stress.
<i>Ey</i>	(float): Young modulus.
<i>b</i>	(float, optional): Strain hardening factor. Defaults to 0.01.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	fy needs to be positive.
<i>NegativeValue</i>	Ey needs to be positive.

Reimplemented in [UniaxialBilinearSteelShape](#).

Definition at line 2326 of file [MaterialModels.py](#).

```

02326     def __init__(self, ID: int, fy, Ey, b = 0.01):
02327         """
02328         Constructor of the class.
02329
02330         @param ID (int): Unique material model ID.
02331         @param fy (float): Yield stress.
02332         @param Ey (float): Young modulus.
02333         @param b (float, optional): Strain hardening factor. Defaults to 0.01.
02334
02335         @exception NegativeValue: ID needs to be a positive integer.
02336         @exception NegativeValue: fy needs to be positive.
02337         @exception NegativeValue: Ey needs to be positive.
02338         """
02339         # Check
02340         if ID < 1: raise NegativeValue()
02341         if fy < 0: raise NegativeValue()
02342         if Ey < 0: raise NegativeValue()
02343
02344         # Arguments
02345         self.ID = ID
02346         self.fy = fy
02347         self.Ey = Ey
02348         self.b = b
02349
02350         # Initialized the parameters that are dependent from others
02351         self.section_name_tag = "None"
02352         self.Initialized = False
02353         self.ReInit()
02354

```

## 7.58.3 Member Function Documentation

### 7.58.3.1 CheckApplicability()

```

def CheckApplicability (
    self )

```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 2423 of file [MaterialModels.py](#).

```
02423     def CheckApplicability(self):
02424         """
02425         Implementation of the homonym abstract method.
02426         See parent class MaterialModels for detailed information.
02427         """
02428         Check = True
02429         # if len(self.wy) == 0 or len(self.wx_top) == 0 or len(self.wx_bottom) == 0:
02430         #     Check = False
02431         #     print("Hypothesis of one bar per corner not fullfilled.")
02432         if not Check:
02433             print("The validity of the equations is not fullfilled.")
02434             print("!!!!!! WARNING !!!!!!! Check material model of Uniaxial Bilinear, ID=", self.ID)
02435             print("")
02436
02437
```

### 7.58.3.2 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 2355 of file [MaterialModels.py](#).

```
02355     def ReInit(self):
02356         """
02357         Implementation of the homonym abstract method.
02358         See parent class DataManagement for detailed information.
02359         """
02360         # Check applicability
02361         self.CheckApplicability()
02362
02363         # Members
02364         self.ey = self.fy / self.Ey
02365         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
02366
02367         # Data storage for loading/saving
02368         self.UpdateStoredData()
02369
02370
```

### 7.58.3.3 ShowInfo()

```
def ShowInfo (
    self,
    plot = False,
    block = False )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

#### Parameters

<i>plot</i>	(bool, optional): Option to show the plot of the material model. Defaults to False.
<i>block</i>	(bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of the program everytime that a plot should pop up). Defaults to False.

Definition at line 2387 of file [MaterialModels.py](#).

```

02387     def ShowInfo(self, plot = False, block = False):
02388         """
02389         Implementation of the homonym abstract method.
02390         See parent class DataManagement for detailed information.
02391
02392         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
02393         False.
02394         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
02395         """
02396         print("")
02397         print("Requested info for Uniaxial Bilinear material model Parameters, ID =
{}".format(self.ID))
02398         print("Section associated: {} ".format(self.section_name_tag))
02399         print('Yielding stress fy = {} MPa'.format(self.fy/MPa_unit))
02400         print('Young modulus Ey = {} MPa'.format(self.Ey/MPa_unit))
02401         print('Maximal elastic strain epsilon y = {}'.format(self.ey))
02402         print('Hardening factor b = {}'.format(self.b))
02403         print("")
02404         if plot:
02405             # Data for plotting
02406             e_pl = 10.0 * self.ey # to show that if continues with this slope
02407             sigma_pl = self.b * self.Ey * e_pl
02408
02409             x_axis = np.array([0.0, self.ey, (self.ey+e_pl)])*100
02410             y_axis = np.array([0.0, self.fy, (self.fy+sigma_pl)])/MPa_unit
02411
02412             fig, ax = plt.subplots()
02413             ax.plot(x_axis, y_axis, 'k-')
02414
02415             ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
02416                   title='Uniaxial Bilinear model for material ID={}'.format(self.ID))
02417             ax.grid()
02418
02419             if block:
02420                 plt.show()
02421
02422

```

### 7.58.3.4 Steel01()

```

def Steel01 (
    self )

```

Generate the material model Steel01 uniaxial bilinear material model.

See `_Steel01` function for more information.

Definition at line 2438 of file [MaterialModels.py](#).

```

02438     def Steel01(self):
02439         """
02440         Generate the material model Steel01 uniaxial bilinear material model.
02441         See _Steel01 function for more information.
02442         """
02443         _Steel01(self.ID, self.fy, self.Ey, self.b)
02444         self.Initialized = True
02445         self.UpdateStoredData()
02446
02447

```

### 7.58.3.5 UpdateStoredData()

```
def UpdateStoredData (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 2372 of file `MaterialModels.py`.

```
02372     def UpdateStoredData(self):
02373         """
02374         Implementation of the homonym abstract method.
02375         See parent class DataManagement for detailed information.
02376         """
02377         self.data = [{"INFO_TYPE", "UniaxialBilinear"}, # Tag for differentiating different data
02378                     ["ID", self.ID],
02379                     ["section_name_tag", self.section_name_tag],
02380                     ["fy", self.fy],
02381                     ["Ey", self.Ey],
02382                     ["ey", self.ey],
02383                     ["b", self.b],
02384                     ["Initialized", self.Initialized]]
02385
02386
```

## 7.58.4 Member Data Documentation

### 7.58.4.1 b

b

Definition at line 2348 of file `MaterialModels.py`.

### 7.58.4.2 data

data

Definition at line 2377 of file `MaterialModels.py`.

### 7.58.4.3 Ey

Ey

Definition at line 2347 of file `MaterialModels.py`.

#### 7.58.4.4 `ey`

`ey`

Definition at line [2364](#) of file [MaterialModels.py](#).

#### 7.58.4.5 `fy`

`fy`

Definition at line [2346](#) of file [MaterialModels.py](#).

#### 7.58.4.6 `ID`

`ID`

Definition at line [2345](#) of file [MaterialModels.py](#).

#### 7.58.4.7 `Initialized`

`Initialized`

Definition at line [2352](#) of file [MaterialModels.py](#).

#### 7.58.4.8 `section_name_tag`

`section_name_tag`

Definition at line [2351](#) of file [MaterialModels.py](#).

The documentation for this class was generated from the following file:

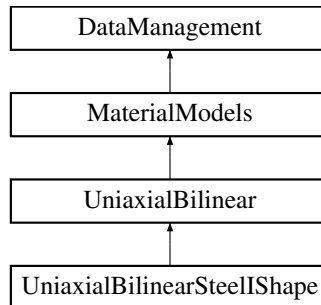
- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)



## 7.59 UniaxialBilinearSteelShape Class Reference

Class that is the children of [UniaxialBilinear](#) and combine the class SteelShape (section) to retrieve the information needed.

Inheritance diagram for UniaxialBilinearSteelShape:



### Public Member Functions

- `def __init__(self, int ID, SteelShape section, b=0.01)`  
*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.59.1 Detailed Description

Class that is the children of [UniaxialBilinear](#) and combine the class SteelShape (section) to retrieve the information needed.

#### Parameters

<a href="#">UniaxialBilinear</a>	Parent class.
----------------------------------	---------------

Definition at line 2448 of file [MaterialModels.py](#).

#### 7.59.2 Constructor & Destructor Documentation

### 7.59.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    SteelIShape section,
    b = 0.01 )
```

Constructor of the class.

It passes the arguments into the parent class to generate the combination of the parent class and the section class `SteelIShape`. The copy of the section passed is stored in the member variable `self.section`.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(SteelIShape): SteelIShape section object.
<i>b</i>	(float, optional): Strain hardening factor. Defaults to 0.01.

Reimplemented from [UniaxialBilinear](#).

Definition at line 2454 of file [MaterialModels.py](#).

```
02454     def __init__(self, ID: int, section: SteelIShape, b=0.01):
02455         """
02456         Constructor of the class. It passes the arguments into the parent class to generate the
02457         combination of the parent class
02458         and the section class SteelIShape.
02459         The copy of the section passed is stored in the member variable self.section.
02460         @param ID (int): Unique material model ID.
02461         @param section (SteelIShape): SteelIShape section object.
02462         @param b (float, optional): Strain hardening factor. Defaults to 0.01.
02463         """
02464         self.section = deepcopy(section)
02465         super().__init__(ID, section.Fy, section.E, b=b)
02466         self.section_name_tag = section.name_tag
02467         self.UpdateStoredData()
02468
02469
```

## 7.59.3 Member Data Documentation

### 7.59.3.1 `section`

`section`

Definition at line 2464 of file [MaterialModels.py](#).

### 7.59.3.2 `section_name_tag`

`section_name_tag`

Definition at line 2466 of file [MaterialModels.py](#).

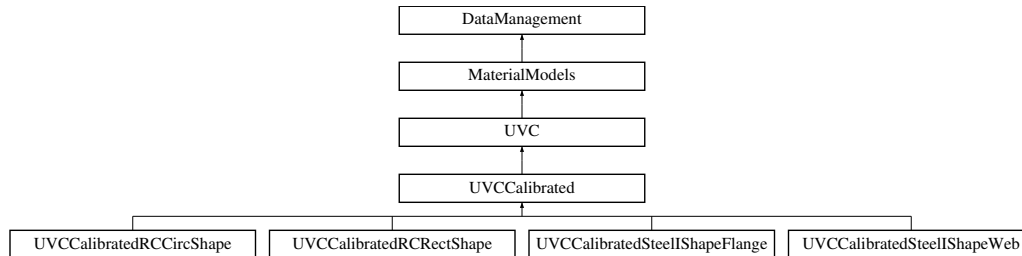
The documentation for this class was generated from the following file:

- `/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py`

## 7.60 UVC Class Reference

Class that stores functions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.

Inheritance diagram for UVC:



### Public Member Functions

- `def __init__(self, int ID, fy, Ey, QInf, b, DInf, a, np.ndarray cK, np.ndarray gammaK)`  
*Constructor of the class.*
- `def CheckApplicability(self)`  
*Implementation of the homonym abstract method.*
- `def Relnit(self)`  
*Implementation of the homonym abstract method.*
- `def ShowInfo(self)`  
*Implementation of the homonym abstract method.*
- `def UpdateStoredData(self)`  
*Implementation of the homonym abstract method.*
- `def UVCuniaxial(self)`  
*Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.*

### Public Attributes

- `a`
- `b`
- `cK`
- `data`
- `DInf`
- `Ey`
- `fy`
- `gammaK`
- `ID`
- `Initialized`
- `N`
- `QInf`
- `section_name_tag`

#### 7.60.1 Detailed Description

Class that stores functions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is UVCuniaxial.

For more information about the how to calibrate the set of parameters, see de Castro e Sousa, Suzuki and Lignos 2020 and Hartloper, de Castro e Sousa and Lignos 2021.

## Parameters

<a href="#">MaterialModels</a>	Parent abstract class.
--------------------------------	------------------------

Definition at line 2629 of file [MaterialModels.py](#).

## 7.60.2 Constructor & Destructor Documentation

### 7.60.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    fy,
    Ey,
    QInf,
    b,
    DInf,
    a,
    np.ndarray cK,
    np.ndarray gammaK )
```

Constructor of the class.

## Parameters

<i>ID</i>	(int): Unique material model ID.
<i>fy</i>	(float): Initial yield stress of the steel material.
<i>Ey</i>	(float): Elastic modulus of the steel material.
<i>QInf</i>	(float): Maximum increase in yield stress due to cyclic hardening (isotropic hardening).
<i>b</i>	(float): Saturation rate of QInf.
<i>DInf</i>	(float): Decrease in the initial yield stress, to neglect the model updates set DInf = 0.
<i>a</i>	(float): Saturation rate of DInf, $a > 0$ . If DInf == 0, then a is arbitrary (but still $a > 0$ ).
<i>cK</i>	(np.ndarray): Array of 1 dimension; each entry is one kinematic hardening parameter associated with one backstress, up to 8 may be specified.
<i>gammaK</i>	(np.ndarray): Array of 1 dimension; each entry is one saturation rate of kinematic hardening associated with one backstress, up to 8 may be specified.

## Exceptions

<i>NegativeValue</i>	ID needs to be a positive integer.
<i>NegativeValue</i>	fy needs to be positive.
<i>NegativeValue</i>	Ey needs to be positive.
<i>NegativeValue</i>	QInf needs to be positive.
<i>NegativeValue</i>	b needs to be positive.
<i>NegativeValue</i>	DInf needs to be positive.
<i>NegativeValue</i>	a needs to be positive.

## Exceptions

<i>WrongArgument</i>	cK can't be empty.
<i>WrongArgument</i>	cK and gammaK have as many entries as the number of backstresses (thus they have the same length).

Reimplemented in [UVCCalibratedRCCircShape](#), [UVCCalibratedRCRectShape](#), [UVCCalibratedSteelIShapeFlange](#), [UVCCalibratedSteelIShapeWeb](#), and [UVCCalibrated](#).

Definition at line 2638 of file [MaterialModels.py](#).

```

02638     def __init__(self, ID: int, fy, Ey, QInf, b, DInf, a, cK: np.ndarray, gammaK: np.ndarray):
02639         """
02640         Constructor of the class.
02641
02642         @param ID (int): Unique material model ID.
02643         @param fy (float): Initial yield stress of the steel material.
02644         @param Ey (float): Elastic modulus of the steel material.
02645         @param QInf (float): Maximum increase in yield stress due to cyclic hardening (isotropic
02646         hardening).
02647         @param b (float): Saturation rate of QInf.
02648         @param DInf (float): Decrease in the initial yield stress, to neglect the model updates set
02649         DInf = 0.
02650         @param a (float): Saturation rate of DInf, a > 0. If DInf == 0, then a is arbitrary (but still
02651         a > 0).
02652         @param cK (np.ndarray): Array of 1 dimension; each entry is one kinematic hardening parameter
02653         associated with one backstress, up to 8 may be specified.
02654         @param gammaK (np.ndarray): Array of 1 dimension; each entry is one saturation rate of
02655         kinematic hardening associated with one backstress, up to 8 may be specified.
02656
02657         @exception NegativeValue: ID needs to be a positive integer.
02658         @exception NegativeValue: fy needs to be positive.
02659         @exception NegativeValue: Ey needs to be positive.
02660         @exception NegativeValue: QInf needs to be positive.
02661         @exception NegativeValue: b needs to be positive.
02662         @exception NegativeValue: DInf needs to be positive.
02663         @exception NegativeValue: a needs to be positive.
02664         @exception WrongArgument: cK can't be empty.
02665         @exception WrongArgument: cK and gammaK have as many entries as the number of backstresses
02666         (thus they have the same length).
02667
02668         """
02669         # Check
02670         if ID < 1: raise NegativeValue()
02671         if fy < 0: raise NegativeValue()
02672         if Ey < 0: raise NegativeValue()
02673         if QInf < 0: raise NegativeValue()
02674         if b < 0: raise NegativeValue()
02675         if DInf < 0: raise NegativeValue()
02676         if a < 0: raise NegativeValue()
02677         if len(cK) == 0: raise WrongArgument()
02678         if len(cK) != len(gammaK): raise WrongArgument()
02679         if len(cK) != 2: print("!!!!!! WARNING !!!!!!! Number of backstresses should be 2 for optimal
02680         performances")
02681         if DInf == 0: print("!!!!!! WARNING !!!!!!! With DInf = 0, the model used is Voce-Chaboche
02682         (VC) not updated (UVC)")
02683
02684         # Arguments
02685         self.ID = ID
02686         self.fy = fy
02687         self.Ey = Ey
02688         self.QInf = QInf
02689         self.b = b
02690         self.DInf = DInf
02691         self.a = a
02692         self.cK = copy(cK)
02693         self.gammaK = copy(gammaK)
02694
02695         # Initialized the parameters that are dependent from others
02696         self.section_name_tag = "None"
02697         self.Initialized = False
02698         self.ReInit()
02699

```

### 7.60.3 Member Function Documentation

### 7.60.3.1 CheckApplicability()

```
def CheckApplicability (
    self )
```

Implementation of the homonym abstract method.

See parent class [MaterialModels](#) for detailed information.

Reimplemented from [MaterialModels](#).

Definition at line 2747 of file [MaterialModels.py](#).

```
02747     def CheckApplicability(self):
02748         """
02749         Implementation of the homonym abstract method.
02750         See parent class MaterialModels for detailed information.
02751         """
02752         Check = True
02753         # No checks
02754         if not Check:
02755             print("The validity of the equations is not fullfilled.")
02756             print("!!!!!! WARNING !!!!!!! Check material model of UVC, ID=", self.ID)
02757             print("")
02758
02759
```

### 7.60.3.2 ReInit()

```
def ReInit (
    self )
```

Implementation of the homonym abstract method.

See parent class [DataManagement](#) for detailed information.

Definition at line 2691 of file [MaterialModels.py](#).

```
02691     def ReInit(self):
02692         """
02693         Implementation of the homonym abstract method.
02694         See parent class DataManagement for detailed information.
02695         """
02696         # Check applicability
02697         self.CheckApplicability()
02698
02699         # Members
02700         self.N = len(self.cK)
02701         if self.section_name_tag != "None": self.section_name_tag = self.section_name_tag + "
(modified)"
02702
02703         # Data storage for loading/saving
02704         self.UpdateStoredData()
02705
02706
```

### 7.60.3.3 ShowInfo()

```
def ShowInfo (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 2728 of file `MaterialModels.py`.

```
02728     def ShowInfo(self):
02729         """
02730         Implementation of the homonym abstract method.
02731         See parent class DataManagement for detailed information.
02732         """
02733         print("")
02734         print("Requested info for UVC material model Parameters, ID = {}".format(self.ID))
02735         print("Section associated: {} ".format(self.section_name_tag))
02736         print("Yield strength fy = {} MPa".format(self.fy/MPa_unit))
02737         print("Young modulus Ey = {} MPa".format(self.Ey/MPa_unit))
02738         print("Isotropic hardening factor QInf = {} MPa and saturation rate b =
{}".format(self.QInf/MPa_unit, self.b))
02739         print("Decrease the initial yield stress DInf = {} MPa and saturation rate a =
{}".format(self.DInf/MPa_unit, self.a))
02740         print("Kinematic hardening vector ({} backstresses) cK = {} MPa".format(self.N,
self.cK/MPa_unit))
02741         print("And associated saturation rate gammaK = {}".format(self.gammaK))
02742         print("")
02743
02744         #TODO: implement plot (too complex for now)
02745
02746
```

### 7.60.3.4 UpdateStoredData()

```
def UpdateStoredData (
    self )
```

Implementation of the homonym abstract method.

See parent class `DataManagement` for detailed information.

Definition at line 2708 of file `MaterialModels.py`.

```
02708     def UpdateStoredData(self):
02709         """
02710         Implementation of the homonym abstract method.
02711         See parent class DataManagement for detailed information.
02712         """
02713         self.data = [{"INFO_TYPE", "UVC"}, # Tag for differentiating different data
02714                     ["ID", self.ID],
02715                     ["section_name_tag", self.section_name_tag],
02716                     ["fy", self.fy],
02717                     ["Ey", self.Ey],
02718                     ["QInf", self.QInf],
02719                     ["b", self.b],
02720                     ["DInf", self.DInf],
02721                     ["a", self.a],
02722                     ["N", self.N],
02723                     ["cK", self.cK],
02724                     ["gammaK", self.gammaK],
02725                     ["Initialized", self.Initialized]]
02726
02727
```

### 7.60.3.5 UVCuniaxial()

```
def UVCuniaxial (
    self )
```

Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.

See `_UVCuniaxial` function for more information.

Definition at line 2760 of file [MaterialModels.py](#).

```
02760     def UVCuniaxial(self):
02761         """
02762         Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.
02763         See _UVCuniaxial function for more information.
02764         """
02765         _UVCuniaxial(self.ID, self.Ey, self.fy, self.QInf, self.b, self.DInf, self.a, self.N, self.cK,
self.gammaK)
02766         self.Initialized = True
02767         self.UpdateStoredData()
02768
02769
```

## 7.60.4 Member Data Documentation

### 7.60.4.1 a

a

Definition at line 2682 of file [MaterialModels.py](#).

### 7.60.4.2 b

b

Definition at line 2680 of file [MaterialModels.py](#).

### 7.60.4.3 cK

cK

Definition at line 2683 of file [MaterialModels.py](#).



#### 7.60.4.4 data

data

Definition at line 2713 of file [MaterialModels.py](#).

#### 7.60.4.5 DInf

DInf

Definition at line 2681 of file [MaterialModels.py](#).

#### 7.60.4.6 Ey

Ey

Definition at line 2678 of file [MaterialModels.py](#).

#### 7.60.4.7 fy

fy

Definition at line 2677 of file [MaterialModels.py](#).

#### 7.60.4.8 gammaK

gammaK

Definition at line 2684 of file [MaterialModels.py](#).

#### 7.60.4.9 ID

ID

Definition at line 2676 of file [MaterialModels.py](#).

#### 7.60.4.10 Initialized

Initialized

Definition at line 2688 of file [MaterialModels.py](#).

#### 7.60.4.11 N

N

Definition at line 2700 of file [MaterialModels.py](#).

#### 7.60.4.12 QInf

QInf

Definition at line 2679 of file [MaterialModels.py](#).

#### 7.60.4.13 section\_name\_tag

section\_name\_tag

Definition at line 2687 of file [MaterialModels.py](#).

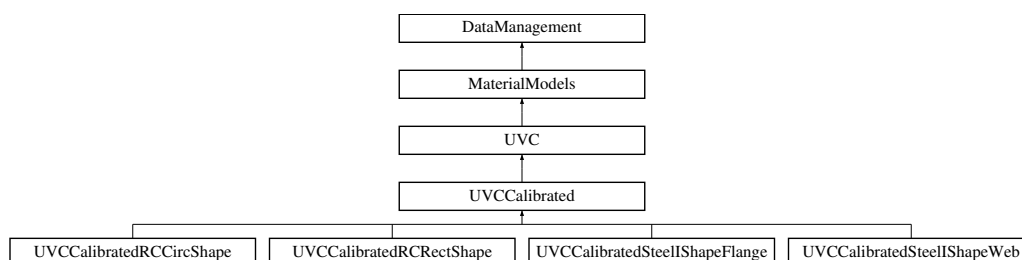
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.61 UVCCalibrated Class Reference

Class that is the children of [UVC](#) that retrieve calibrated parameters from UVC\_calibrated\_parameters.txt.

Inheritance diagram for UVCCalibrated:



## Public Member Functions

- `def __init__ (self, int ID, str calibration, fy=-1, E=-1)  
Constructor of the class.`

## Public Attributes

- [calibration](#)

### 7.61.1 Detailed Description

Class that is the children of [UVC](#) that retrieve calibrated parameters from `UVC_calibrated_parameters.txt`.

The file text can be modified by adding more calibrated parameters.

#### Parameters

<a href="#">UVC</a>	Parent class.
---------------------	---------------

Definition at line [2770](#) of file [MaterialModels.py](#).

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    int ID,
    str calibration,
    fy = -1,
    E = -1 )
```

Constructor of the class.

It retrieve the parameters from `UVC_calibrated_parameters.txt` and pass them in the parent class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>calibration</i>	(str): Label of the calibration parameter set. The options are:

7.61.3 'S355J2\_25mm\_plate' \n

7.61.4 'S355J2\_50mm\_plate' \n

7.61.5 'S355J2\_HEB500\_flange' \n

7.61.6 'S355J2\_HEB500\_web' \n

7.61.7 'S460NL\_25mm\_plate' \n

7.61.8 'S690QL\_25mm\_plate' \n

7.61.9 'A992Gr50\_W14X82\_web' \n

7.61.10 'A992Gr50\_W14X82\_flange' \n

7.61.11 'A500GrB\_HSS305X16' \n

7.61.12 'BCP325\_22mm\_plate' \n

7.61.13 'BCR295\_HSS350X22' \n

7.61.14 'HYP400\_27mm\_plate' \n

#### Parameters

<i>fy</i>	(float, optional): Yield strength. Defaults to -1, e.g. taken equal to the one given in the calibration parameter set.
<i>E</i>	(float, optional): Young modulus. Defaults to -1, e.g. taken equal to the one given in the calibration parameter set.

#### Exceptions

<i>NegativeValue</i>	<i>fy</i> needs to be positive if different from -1.
<i>NegativeValue</i>	<i>E</i> needs to be positive if different from -1.
<i>NameError</i>	calibration needs to be equal to the label of one of the set of calibrated parameters.

Reimplemented from [UVC](#).

Reimplemented in [UVCCalibratedRCCircShape](#), [UVCCalibratedRCRectShape](#), [UVCCalibratedSteelIShapeFlange](#), and [UVCCalibratedSteelIShapeWeb](#).

Definition at line 2777 of file [MaterialModels.py](#).

```
02777     def __init__(self, ID: int, calibration: str, fy = -1, E = -1):
02778         """
02779         Constructor of the class. It retrieve the parameters from UVC_calibrated_parameters.txt and
            pass them in the parent class.
```

```

02780
02781     @param ID (int): Unique material model ID.
02782     @param calibration (str): Label of the calibration parameter set. The options are: \n
02783     # 'S355J2_25mm_plate' \n
02784     # 'S355J2_50mm_plate' \n
02785     # 'S355J2_HEB500_flange' \n
02786     # 'S355J2_HEB500_web' \n
02787     # 'S460NL_25mm_plate' \n
02788     # 'S690QL_25mm_plate' \n
02789     # 'A992Gr50_W14X82_web' \n
02790     # 'A992Gr50_W14X82_flange' \n
02791     # 'A500GrB_HSS305X16' \n
02792     # 'BCP325_22mm_plate' \n
02793     # 'BCR295_HSS350X22' \n
02794     # 'HYP400_27mm_plate' \n
02795     @param fy (float, optional): Yield strength. Defaults to -1, e.g. taken equal to the one given
in the calibration parameter set.
02796     @param E (float, optional): Young modulus. Defaults to -1, e.g. taken equal to the one given
in the calibration parameter set.
02797
02798     @exception NegativeValue: fy needs to be positive if different from -1.
02799     @exception NegativeValue: E needs to be positive if different from -1.
02800     @exception NameError: calibration needs to be equal to the label of one of the set of
calibrated parameters.
02801     """
02802     if fy != -1 and fy < 0: raise NegativeValue()
02803     if E != -1 and E < 0: raise NegativeValue()
02804
02805     self.calibration = calibration
02806
02807     # Structure of the data to be stored
02808     names = ["Material", "Ey", "fy", "QInf", "b", "DInf", "a", "C1", "gamma1", "C2", "gamma2"]
02809     # Get the data
02810     __location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
02811     UVC_data = np.genfromtxt(os.path.join(__location__, 'UVC_calibrated_parameters.txt'),
dtype=None, skip_header=1, names = names, encoding='ascii', delimiter='\t')
02812     # Define the index (with the location of the correct set of parameters)
02813     index = UVC_data["Material"] == calibration
02814     fy = UVC_data["fy"][index][0]*MPa_unit if fy == -1 else fy
02815     E = UVC_data["Ey"][index][0]*GPa_unit if E == -1 else E
02816     # Check
02817     if not index.any(): raise NameError("No calibrated parameters with that name. Note that there
are no spaces in the label.")
02818
02819     # Assign arguments value
02820     super().__init__(ID, fy, E, UVC_data["QInf"][index][0]*MPa_unit, UVC_data["b"][index][0],
UVC_data["DInf"][index][0]*MPa_unit, UVC_data["a"][index][0],
02821 np.array([UVC_data["C1"][index][0], UVC_data["C2"][index][0]])*MPa_unit,
02822 np.array([UVC_data["gamma1"][index][0], UVC_data["gamma2"][index][0]]))
02823
02824
02825

```

## 7.61.15 Member Data Documentation

### 7.61.15.1 calibration

calibration

Definition at line 2805 of file [MaterialModels.py](#).

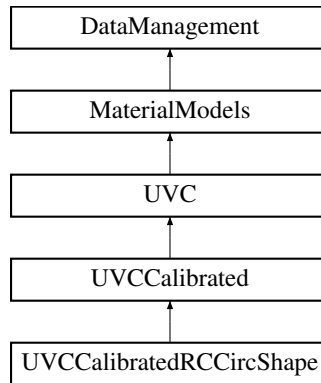
The documentation for this class was generated from the following file:

- [/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py](#)

## 7.62 UVCCalibratedRCCircShape Class Reference

Class that is the children of [UVCCalibrated](#) and combine the class RCCircShape (section) to retrieve the information needed.

Inheritance diagram for UVCCalibratedRCCircShape:



### Public Member Functions

- `def __init__(self, int ID, RCCircShape section, calibration='S460NL_25mm_plate')`  
*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.62.1 Detailed Description

Class that is the children of [UVCCalibrated](#) and combine the class RCCircShape (section) to retrieve the information needed.

#### Parameters

<a href="#">UVCCalibrated</a>	Parent class.
-------------------------------	---------------

Definition at line 2847 of file [MaterialModels.py](#).

#### 7.62.2 Constructor & Destructor Documentation

### 7.62.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RCCircShape section,
    calibration = 'S460NL_25mm_plate' )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(RCCircShape): RCCircShape section object.
<i>calibration</i>	(str, optional): Label of the calibration parameter set. The options are listed in <a href="#">UVCCalibrated</a> . Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material properties.

Reimplemented from [UVCCalibrated](#).

Definition at line 2853 of file [MaterialModels.py](#).

```
02853     def __init__(self, ID: int, section: RCCircShape, calibration = 'S460NL_25mm_plate'):
02854         """
02855         Constructor of the class.
02856
02857         @param ID (int): Unique material model ID.
02858         @param section (RCCircShape): RCCircShape section object.
02859         @param calibration (str, optional): Label of the calibration parameter set. The options are
02860         listed in UVCCalibrated. Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material
02861         properties.
02862         """
02863         self.section = deepcopy(section)
02864         super().__init__(ID, calibration, section.fy, section.Ey)
02865         self.section_name_tag = section.name_tag
02866         self.UpdateStoredData()
02867
```

## 7.62.3 Member Data Documentation

### 7.62.3.1 `section`

`section`

Definition at line 2862 of file [MaterialModels.py](#).

### 7.62.3.2 `section_name_tag`

`section_name_tag`

Definition at line 2864 of file [MaterialModels.py](#).

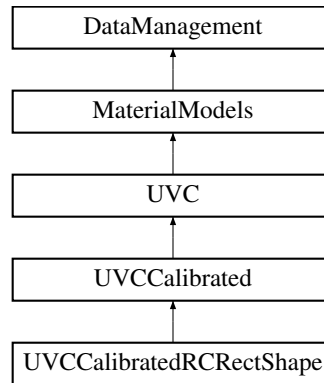
The documentation for this class was generated from the following file:

- `/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py`

## 7.63 UVCCalibratedRRectShape Class Reference

Class that is the children of [UVCCalibrated](#) and combines the class RRectShape (section) to retrieve the information needed.

Inheritance diagram for UVCCalibratedRRectShape:



### Public Member Functions

- `def __init__ (self, int ID, RRectShape section, calibration='S460NL_25mm_plate')`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.63.1 Detailed Description

Class that is the children of [UVCCalibrated](#) and combines the class RRectShape (section) to retrieve the information needed.

#### Parameters

<a href="#">UVCCalibrated</a>	Parent class.
-------------------------------	---------------

Definition at line [2826](#) of file [MaterialModels.py](#).

#### 7.63.2 Constructor & Destructor Documentation



### 7.63.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    RCRectShape section,
    calibration = 'S460NL_25mm_plate' )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(RCRectShape): RCRectShape section object.
<i>calibration</i>	(str): Label of the calibration parameter set. The options are listed in <a href="#">UVCCalibrated</a> . Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material properties.

Reimplemented from [UVCCalibrated](#).

Definition at line 2832 of file [MaterialModels.py](#).

```
02832     def __init__(self, ID: int, section: RCRectShape, calibration = 'S460NL_25mm_plate'):
02833         """
02834         Constructor of the class.
02835
02836         @param ID (int): Unique material model ID.
02837         @param section (RCRectShape): RCRectShape section object.
02838         @param calibration (str): Label of the calibration parameter set. The options are listed in
02839         UVCCalibrated.
02839         Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material
02840         properties.
02840         """
02841         self.section = deepcopy(section)
02842         super().__init__(ID, calibration, section.fy, section.Ey)
02843         self.section_name_tag = section.name_tag
02844         self.UpdateStoredData()
02845
02846
```

## 7.63.3 Member Data Documentation

### 7.63.3.1 `section`

`section`

Definition at line 2841 of file [MaterialModels.py](#).

### 7.63.3.2 `section_name_tag`

`section_name_tag`

Definition at line 2843 of file [MaterialModels.py](#).

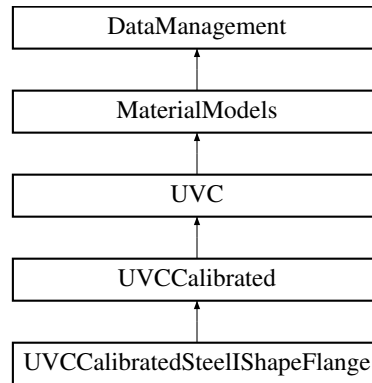
The documentation for this class was generated from the following file:

- `/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py`

## 7.64 UVCCalibratedSteelIShapeFlange Class Reference

Class that is the children of [UVCCalibrated](#) and combine the class [SteelIShape](#) (section) to retrieve the information needed for the material model of the flange (often used to the entire section).

Inheritance diagram for UVCCalibratedSteelIShapeFlange:



### Public Member Functions

- `def __init__ (self, int ID, SteelIShape section, calibration='S355J2_HEB500_flange')`  
*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.64.1 Detailed Description

Class that is the children of [UVCCalibrated](#) and combine the class [SteelIShape](#) (section) to retrieve the information needed for the material model of the flange (often used to the entire section).

Parameters

<a href="#">UVCCalibrated</a>	Parent class.
-------------------------------	---------------

Definition at line [2868](#) of file [MaterialModels.py](#).

#### 7.64.2 Constructor & Destructor Documentation

### 7.64.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    SteelIShape section,
    calibration = 'S355J2_HEB500_flange' )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(SteelIShape): SteelIShape section object.
<i>calibration</i>	(str, optional): Label of the calibration parameter set. The options are listed in <a href="#">UVCCalibrated</a> . Defaults to 'S355J2_HEB500_flange'. Change it accordingly to the steel rebars material properties.

Reimplemented from [UVCCalibrated](#).

Definition at line 2875 of file [MaterialModels.py](#).

```
02875     def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_flange'):
02876         """
02877         Constructor of the class.
02878
02879         @param ID (int): Unique material model ID.
02880         @param section (SteelIShape): SteelIShape section object.
02881         @param calibration (str, optional): Label of the calibration parameter set. The options are
02882         listed in UVCCalibrated. Defaults to 'S355J2_HEB500_flange'. Change it accordingly to the steel rebars material
02883         properties.
02884         """
02885         self.section = deepcopy(section)
02886         super().__init__(ID, calibration, section.Fy, section.E)
02887         self.section_name_tag = section.name_tag
02888         self.UpdateStoredData()
02889
```

## 7.64.3 Member Data Documentation

### 7.64.3.1 `section`

`section`

Definition at line 2884 of file [MaterialModels.py](#).

### 7.64.3.2 `section_name_tag`

`section_name_tag`

Definition at line 2886 of file [MaterialModels.py](#).

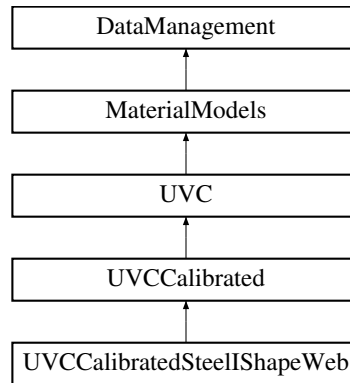
The documentation for this class was generated from the following file:

- `/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py`

## 7.65 UVCCalibratedSteelShapeWeb Class Reference

Class that is the children of [UVCCalibrated](#) and combine the class SteelShape (section) to retrieve the information needed for the material model of the web.

Inheritance diagram for UVCCalibratedSteelShapeWeb:



### Public Member Functions

- `def __init__ (self, int ID, SteelShape section, calibration='S355J2_HEB500_web')`

*Constructor of the class.*

### Public Attributes

- [section](#)
- [section\\_name\\_tag](#)

#### 7.65.1 Detailed Description

Class that is the children of [UVCCalibrated](#) and combine the class SteelShape (section) to retrieve the information needed for the material model of the web.

#### Parameters

<a href="#">UVCCalibrated</a>	Parent class.
-------------------------------	---------------

Definition at line [2890](#) of file [MaterialModels.py](#).

#### 7.65.2 Constructor & Destructor Documentation

### 7.65.2.1 `__init__()`

```
def __init__ (
    self,
    int ID,
    SteelIShape section,
    calibration = 'S355J2_HEB500_web' )
```

Constructor of the class.

#### Parameters

<i>ID</i>	(int): Unique material model ID.
<i>section</i>	(SteelIShape): SteelIShape section object.
<i>calibration</i>	(str, optional): Label of the calibration parameter set. The options are listed in <a href="#">UVCCalibrated</a> . Defaults to 'S355J2_HEB500_web'. Change it accordingly to the steel rebars material properties.

Reimplemented from [UVCCalibrated](#).

Definition at line 2897 of file [MaterialModels.py](#).

```
02897     def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_web'):
02898         """
02899         Constructor of the class.
02900
02901         @param ID (int): Unique material model ID.
02902         @param section (SteelIShape): SteelIShape section object.
02903         @param calibration (str, optional): Label of the calibration parameter set. The options are
02904         listed in UVCCalibrated.
02905         Defaults to 'S355J2_HEB500_web'. Change it accordingly to the steel rebars material
02906         properties.
02907         """
02908         self.section = deepcopy(section)
02909         super().__init__(ID, calibration, section.Fy_web, section.E)
02910         self.section_name_tag = section.name_tag
02911         self.UpdateStoredData()
02912     # Public functions
```

## 7.65.3 Member Data Documentation

### 7.65.3.1 `section`

`section`

Definition at line 2906 of file [MaterialModels.py](#).

### 7.65.3.2 `section_name_tag`

`section_name_tag`

Definition at line 2908 of file [MaterialModels.py](#).

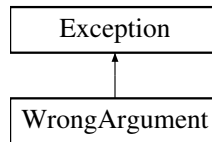
The documentation for this class was generated from the following file:

- `/media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py`

## 7.66 WrongArgument Class Reference

Exception class for the "input of a wrong argument" error.

Inheritance diagram for WrongArgument:



### 7.66.1 Detailed Description

Exception class for the "input of a wrong argument" error.

Definition at line 11 of file [ErrorHandling.py](#).

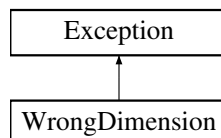
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.67 WrongDimension Class Reference

Exception class for the "wrong array dimensions" error.

Inheritance diagram for WrongDimension:



### 7.67.1 Detailed Description

Exception class for the "wrong array dimensions" error.

Definition at line 26 of file [ErrorHandling.py](#).

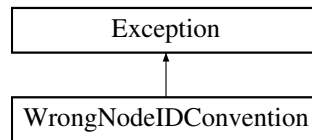
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.68 WrongNodeIDConvention Class Reference

Exception class for the "wrong node ID convention definition" error.

Inheritance diagram for WrongNodeIDConvention:



### Public Member Functions

- `def __init__(self, node)`

### Public Attributes

- `node`

#### 7.68.1 Detailed Description

Exception class for the "wrong node ID convention definition" error.

Definition at line 41 of file [ErrorHandling.py](#).

#### 7.68.2 Constructor & Destructor Documentation

##### 7.68.2.1 \_\_init\_\_()

```
def __init__ (  
    self,  
    node )
```

Definition at line 44 of file [ErrorHandling.py](#).

```
00044     def __init__(self, node):  
00045         self.node = node  
00046
```

##### 7.68.3 Member Data Documentation

### 7.68.3.1 node

node

Definition at line 45 of file [ErrorHandling.py](#).

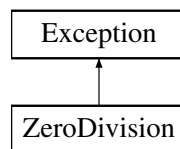
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.69 ZeroDivision Class Reference

Exception class for the "zero division" error.

Inheritance diagram for ZeroDivision:



### 7.69.1 Detailed Description

Exception class for the "zero division" error.

Definition at line 6 of file [ErrorHandling.py](#).

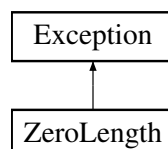
The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)

## 7.70 ZeroLength Class Reference

Exception class for the "zero length element (non intentional)" error.

Inheritance diagram for ZeroLength:



### Public Member Functions

- `def \_\_init\_\_(self, element)`



## Public Attributes

- [element](#)

### 7.70.1 Detailed Description

Exception class for the "zero length element (non intentional)" error.

Definition at line 52 of file [ErrorHandling.py](#).

### 7.70.2 Constructor & Destructor Documentation

#### 7.70.2.1 `__init__()`

```
def __init__ (
    self,
    element )
```

Definition at line 55 of file [ErrorHandling.py](#).

```
00055     def __init__(self, element):
00056         self.element = element
```

### 7.70.3 Member Data Documentation

#### 7.70.3.1 `element`

`element`

Definition at line 56 of file [ErrorHandling.py](#).

The documentation for this class was generated from the following file:

- [/media/carminc/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py](#)



## Chapter 8

# File Documentation

### 8.1 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/AnalysisAndPostProcessing.py File Reference

#### Classes

- class [Analysis](#)  
*Class dedicated to the analysis of the OpenSeesPy model.*

#### Namespaces

- namespace [AnalysisAndPostProcessing](#)  
*Module with pre-made analysis and postprocessing functions.*

### 8.2 AnalysisAndPostProcessing.py

[Go to the documentation of this file.](#)

```
00001 """Module with pre-made analysis and postprocessing functions. \n
00002 Carmine Schipani, 2021
00003 """
00004
00005 from openseespy.opensees import *
00006 import matplotlib.pyplot as plt
00007 import numpy as np
00008 import os
00009 import openseespy.postprocessing.Get_Rendering as opsplt
00010 from OpenSeesPyAssistant.ErrorHandling import *
00011 from OpenSeesPyAssistant.Units import *
00012 from OpenSeesPyAssistant.Constants import *
00013 from OpenSeesPyAssistant.FunctionalFeatures import *
00014
00015
00016 class Analysis():
00017     """Class dedicated to the analysis of the OpenSeesPy model. The Gravity method should be run first
00018     to perform the Load-control analysis (apply the vertical load). If no vertical load, this method can
00019     be omitted. \n
00020     Then only one of the Displacement-control (Pushover or LoadingProtocol) or Load-control
00021     (LateralForce) analysis can ran. \n
00022     After the analysis reach convergence in the last step, for the postprocessing, the DeformedShape
00023     method can be used to see the final deformed shape and the animation of the entire loading protocol;
00024     the FiberResponse method can be used to see the animation of the same fiber section recorded
00025     during the analysis (strain and/or stress).
00026     """
00027     def __init__(self, data_dir: str, name_ODB: str, algo = "KrylovNewton", test_type =
00028         "NormDispIncr", test_opt = 0, max_iter = MAX_ITER, tol = TOL, allow_smaller_step = False):
```

```

00023     """
00024     The constructor of the class.
00025
00026     @param data_dir (str): Directory in which the results from the analysis will be stored. Use
the recorders (from OpenSeesPy) or the Record method from MemberModel.
00027     @param name_ODB (str): Name for the folder in which the data for the animations and the fibers
are stored.
00028     @param algo (str, optional): Type of algorithm chosen for the analysis. It determines how to
construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the
non-linear equation.
00029     For more information on the available types, see the OpenSeesPy documentation. Defaults to
"KrylovNewton".
00030     @param test_type (str, optional): Type of test chosen for the analysis. It determines how to
construct a ConvergenceTest object.
00031     Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if
convergence has been achieved at the end of an iteration step.
00032     For more information on the available types, see the OpenSeesPy documentation. Defaults to
"NormDispIncr".
00033     @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the
iteration
00034     (for example: 0 print nothing and 2 print information on norms and number of iterations at
end of successful test).
00035     For more information, see the OpenSeesPy documentation. Defaults to 0.
00036     @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER
(from Constants Module).
00037     @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to
TOL (from Constants Module).
00038     @param allow_smaller_step (bool, optional): Allow smaller steps in the displacement-control
analysis. Defaults to False.
00039
00040     @exception NegativeValue: The argument max_iter should be positive.
00041     @exception NegativeValue: The argument tol should be positive.
00042     """
00043     if max_iter < 0: raise NegativeValue()
00044     if tol < 0: raise NegativeValue()
00045     if not os.path.exists(data_dir):
00046         print("Folder {} not found in this directory; creating one".format(data_dir))
00047         os.makedirs(data_dir)
00048
00049     self.data_dir = data_dir
00050     self.name_ODB = name_ODB
00051     self.algo = algo
00052     self.test_type = test_type
00053     self.tol = tol
00054     self.test_opt = test_opt
00055     self.max_iter = max_iter
00056     self.allow_smaller_step = allow_smaller_step
00057     self.load_case = "None"
00058
00059
00060     def Gravity(self, loaded_nodes: list, Fy: list, timeSeries_ID: int, pattern_ID: int, n_step = 10,
timeSeries_type = "Linear", pattern_type = "Plain",
00061     constraints_type = "Plain", numberer_type = "RCM", system_type = "BandGeneral", analysis_type
= "Static", show_plot = False):
00062     """
00063     Method to perform the gravity analysis with vertical loadings (load-control).
00064     It can be used before calling the Pushover or LoadingProtocol methods that perform the actual
analysis. If no vertical loadings present, this method can be avoided.
00065
00066     @param loaded_nodes (list): List of nodes that are loaded by the forces in Fy. The first
node will be recorded (thus usually should be in the roof).
00067     @param Fy (list): List of vertical loadings (negative is toward the ground, thus compression;
see global coordinate system).
00068     @param timeSeries_ID (int): ID of the timeseries.
00069     @param pattern_ID (int): ID of the pattern.
00070     @param n_step (int, optional): Number of steps used to during the analysis to reach the
objective state (with 100% vertical loadings imposed). Defaults to 10.
00071     @param timeSeries_type (str, optional): Type of timeseries chosen.
00072     For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00073     @param pattern_type (str, optional): Type of pattern chosen.
00074     For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00075     @param constraints_type (str, optional): Type of constraints chosen. It determines how the
constraint equations are enforced in the analysis.
00076     For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00077     @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
between equation numbers and degrees-of-freedom.
00078     For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00079     @param system_type (str, optional): Type of system of equations chosen. It determines how to
construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
analysis.
00080     For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
00081     @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
the Analysis object, which defines what type of analysis is to be performed.
00082     For more information, see the OpenSeesPy documentation. Defaults to "Static".
00083     @param show_plot (bool, optional): Option to show the 'vertical displacement vs. vertical
loading' curve after the analysis. Defaults to False.
00084

```

```

00085         @exception WrongDimension: The dimension of the loaded_nodes and Fy arguments needs to be the
same.
00086         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00087         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00088         """
00089         if len(loaded_nodes) != len(Fy): raise WrongDimension()
00090         if timeSeries_ID < 1: raise NegativeValue()
00091         if pattern_ID < 1: raise NegativeValue()
00092
00093         # for mass defined: opsplt.createODB(self.name_ODB, "Gravity", Nmodes = nEigen);
00094         # for tracking gravity with ODB: opsplt.createODB(self.name_ODB, "Gravity");
00095
00096         # Create load pattern
00097         timeSeries(timeSeries_type, timeSeries_ID)
00098         pattern(pattern_type, timeSeries_ID, pattern_ID)
00099         for ii, node_ID in enumerate(loaded_nodes):
00100             load(node_ID, 0.0, Fy[ii], 0.0) # load(IDNode, Fx, Fy, Mz)
00101             DGravity = 1.0/n_step # load increment
00102
00103         # Set up analysis options
00104         constraints(constraints_type) # how it handles boundary conditions
00105         numberer(numberer_type) # renumber dof's to minimize band-width (optimization)
00106         system(system_type) # how to store and solve the system of equations in the
analysis
00107                                     # For static model, BandGeneral, for transient and/or big
model, UmfPack
00108         integrator("LoadControl", DGravity) # LoadControl and DisplacementControl only with static
model, linear TimeSeries w/ factor of 1
00109                                     # Newmark used for transient model
00110         algorithm("Newton") # placeholder
00111         analysis(analysis_type) # define type of analysis: static for pushover
00112
00113         # Analysis
00114         dataG = np.zeros((n_step+1,2))
00115         print("")
00116         print("Gravity analysis starts")
00117         for iteration in range(n_step):
00118             convergence = self.__LoadCtrlLoop__LoadCtrlLoop(DGravity, iteration,
00119                 self.algoalgo, self.test_ttest_type, self.toltol, self.test_opttest_opt,
self.max_itermax_iter)
00120             if convergence != 0: break
00121             dataG[iteration+1,0] = nodeDisp(loaded_nodes[0], 2)/mm_unit
00122             dataG[iteration+1,1] = getLoadFactor(pattern_ID)*Fy[0]/kN_unit
00123
00124             if show_plot:
00125                 plt.plot(dataG[:,0], dataG[:,1])
00126                 plt.xlabel('Vertical Displacement [mm]')
00127                 plt.ylabel('Vertical Load [kN]')
00128                 plt.title('Gravity curve')
00129                 plt.show()
00130
00131             loadConst("-time", 0.0)
00132
00133         print("")
00134         print("Gravity complete")
00135
00136
00137         def LateralForce(self, loaded_nodes: list, Fx: list, timeSeries_ID: int, pattern_ID: int, n_step =
1000, fiber_ID_analysed = -1, fiber_section = 1,
00138             timeSeries_type = "Linear", pattern_type = "Plain", constraints_type = "Plain", numberer_type
= "RCM", system_type = "BandGeneral", analysis_type = "Static",
00139             show_plot = True, block = False):
00140             """
00141             Method to perform the lateral force analysis with lateral loading (load-control).
00142             If this method is called, the LoadingProtocol and Pushover methods should be avoided.
00143
00144             @param loaded_nodes (list): List of nodes that are loaded by the the forces in Fx. The first
node will be recorded (thus usually should be in the roof).
00145             @param Fx (list): List of horizontal loadings (negative is toward left; see global coordinate
system).
00146             @param timeSeries_ID (int): ID of the timeseries.
00147             @param pattern_ID (int): ID of the pattern.
00148             @param n_step (int, optional): Number of steps used to during the analysis to reach the
objective state (with 100% horizontal loadings imposed). Defaults to 1000.
00149             @param fiber_ID_analysed (int, optional): The ID of the analysed fiber. If fibers are present
in the model and the user wants to save ODB data
00150             (to use in the post-processing with for example FiberResponse), assign to this argument
the ID of the fiber chosen.
00151             -1 will ignore the storage of data for fibers. Defaults to -1.
00152             @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00153             If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00154             @param timeSeries_type (str, optional): Type of timeseries chosen.
00155             For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00156             @param pattern_type (str, optional): Type of pattern chosen.
00157             For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00158             @param constraints_type (str, optional): Type of constraints chosen. It determines how the
constraint equations are enforced in the analysis.

```

```

00159         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00160         @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
between equation numbers and degrees-of-freedom.
00161         For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00162         @param system_type (str, optional): Type of system of equations chosen. It determines how to
construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
analysis.
00163         For more information, see the OpenSeesPy documentation. Defaults to "BandGeneral".
00164         @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
the Analysis object, which defines what type of analysis is to be performed.
00165         For more information, see the OpenSeesPy documentation. Defaults to "Static".
00166         @param show_plot (bool, optional): Option to show the 'Horizontal displacement vs. Horizontal
loading' curve after the analysis. Defaults to True.
00167         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00168
00169         @exception WrongDimension: The dimension of the loaded_nodes and Fx arguments needs to be the
same.
00170         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00171         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00172         @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer.
00173         """
00174         if len(loaded_nodes) != len(Fx): raise WrongDimension()
00175         if timeSeries_ID < 1: raise NegativeValue()
00176         if pattern_ID < 1: raise NegativeValue()
00177         if fiber_ID_analysed != -1 and fiber_ID_analysed < 1: raise NegativeValue()
00178
00179         # for mass defined: opsplt.createODB(self.name_ODB, "LateralForce", Nmodes = nEigen);
00180         opsplt.createODB(self.name_ODBname_ODB, "LateralForce");
00181         if fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODBname_ODB, "LateralForce",
fiber_ID_analysed, fiber_section)
00182
00183         # Create load pattern
00184         timeSeries(timeSeries_type, timeSeries_ID)
00185         pattern(pattern_type, timeSeries_ID, pattern_ID)
00186         for ii, node_ID in enumerate(loaded_nodes):
00187             load(node_ID, Fx[ii], 0.0, 0.0) # load(IDNode, Fx, Fy, Mz)
00188             force = 1.0/n_step # load increment
00189
00190         # Set up analysis options
00191         constraints(constraints_type) # how it handles boundary conditions
00192         numberer(numberer_type) # renumber dof's to minimize band-width (optimization)
00193         system(system_type) # how to store and solve the system of equations in the
analysis
00194
00195         # For static model, BandGeneral, for transient and/or big
model, UmfPack
00196         integrator("LoadControl", force) # LoadControl and DisplacementControl only with static model,
linear TimeSeries w/ factor of 1
00197
00198         # Newmark used for transient model
algorithm("Newton") # placeholder
00199         analysis(analysis_type) # define type of analysis: static for pushover
00200
00201         # Analysis
dataLF = np.zeros((n_step+1,2))
00202         print("")
00203         print("Lateral Force analysis starts")
00204         for iteration in range(n_step):
00205             convergence = self.__LoadCtrlLoop__LoadCtrlLoop(force, iteration,
self.algoalgo, self.test_typedtest_type, self.toltol, self.test_opttest_opt,
00206 self.max_itermax_iter)
00207             if convergence != 0: break
00208             dataLF[iteration+1,0] = nodeDisp(loaded_nodes[0], 1)/mm_unit
00209             dataLF[iteration+1,1] = getLoadFactor(pattern_ID)*Fx[0]/kN_unit
00210
00211         if show_plot:
00212             plt.plot(dataLF[:,0], dataLF[:,1])
00213             plt.xlabel('Lateral Displacement [mm]')
00214             plt.ylabel('Lateral Load [kN]')
00215             plt.title('Lateral force curve')
00216             if block:
00217                 plt.show()
00218
00219         loadConst("-time", 0.0)
00220
00221         print("")
00222         print("Lateral force complete")
00223         self.load_caseload_case = "LateralForce"
00224
00225         wipe()
00226
00227
00228         def Pushover(self, CtrlNode: int, Dmax, Dincr, timeSeries_ID: int, pattern_ID: int, Fx =
1*kN_unit, ele_fiber_ID_analysed = -1, fiber_section = 1,
00229             timeSeries_type = "Linear", pattern_type = "Plain", constraints_type = "Plain", numberer_type
= "RCM", system_type = "UmfPack", analysis_type = "Static",
00230             show_plot = True, block = False):
00231             """

```

```

00232         Method to perform a pushover analysis (displacement-control). If this method is called, the
LoadingProtocol and LateralForce methods should be avoided.
00233
00234         @param CtrlNode (int): The node that will be used to impose the displacement Dmax of the
pushover analysis.
00235         If the show_plot option is True, the curve displayed follows this node.
00236         @param Dmax (float): The imposed displacement.
00237         @param Dincr (float): The incremental displacement to reach Dmax. To converge, it should be
small enough (1000 times smaller of Dmax).
00238         @param timeSeries_ID (int): ID of the timeseries.
00239         @param pattern_ID (int): ID of the pattern.
00240         @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for
convergence reasons and it can be arbitrarily small.
00241         Defaults to 1*kN_unit.
00242         @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If
fibers are present in the model and the user wants to save ODB data
00243         (to use in the post-processing with for example FiberResponse), assign to this argument
the ID of the element with fibers chosen.
00244         -1 will ignore the storage of data for fibers. Defaults to -1.
00245         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00246         If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00247         @param timeSeries_type (str, optional): Type of timeseries chosen.
00248         For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00249         @param pattern_type (str, optional): Type of pattern chosen.
00250         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00251         @param constraints_type (str, optional): Type of constraints chosen. It detemines how the
constraint equations are enforced in the analysis.
00252         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00253         @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
between equation numbers and degrees-of-freedom.
00254         For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00255         @param system_type (str, optional): Type of system of equations chosen. It determines how to
construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
analysis.
00256         For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
00257         @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
the Analysis object, which defines what type of analysis is to be performed.
00258         For more information, see the OpenSeesPy documentation. Defaults to "Static".
00259         @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral
loading' curve after the analysis. Defaults to True.
00260         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00261
00262         @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
00263         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00264         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00265         @exception NegativeValue: The ID of ele_fiber_ID_analysed needs to be a positive integer if is
different from -1.
00266         """
00267         if CtrlNode < 1: raise NegativeValue()
00268         if timeSeries_ID < 1: raise NegativeValue()
00269         if pattern_ID < 1: raise NegativeValue()
00270         if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()
00271
00272         # for mass defined: opsplt.createODB(self.name_ODB, "Pushover", Nmodes = nEigen);
00273         opsplt.createODB(self.name_ODBname_ODB, "Pushover");
00274         if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODBname_ODB, "Pushover",
ele_fiber_ID_analysed, fiber_section)
00275
00276         # Create load pattern
00277         timeSeries(timeSeries_type, timeSeries_ID)
00278         pattern(pattern_type, timeSeries_ID, pattern_ID)
00279         load(CtrlNode, Fx, 0.0, 0.0) # load(IDNode, Fx, Fy, Mz)
00280         Nsteps = int(abs(Dmax/Dincr)) # number of pushover analysis steps
00281
00282         # Set up analysis options
00283         constraints(constraints_type) # how it handles boundary conditions
00284         numberer(numberer_type) # renumber dof's to minimize band-width (optimization)
00285         system(system_type) # how to store and solve the system of equations in the
analysis
00286
00287         # For static model, BandGeneral, for transient and/or big
model, UmfPack
00287         integrator("LoadControl", 1) # placeholder
00288         algorithm("Newton") # placeholder
00289         analysis(analysis_type) # define type of analysis: static for pushover
00290
00291         # Analysis
00292         dataPO = np.zeros((Nsteps+1,2))
00293         next_step = 0
00294         print("")
00295         print("Pushover analysis starts")
00296         for iteration in range(Nsteps):
00297             next_step = ProgressingPercentage(Nsteps, iteration, next_step)
00298             convergence = self.__LatDispCtrlLoop__LatDispCtrlLoop(CtrlNode, Dincr, iteration,
self.algoalgo, self.test_typedtest_type, self.toltol, self.test_opttest_opt,
self.max_itermax_iter, self.allow_smaller_stepallow_smaller_step)
00299             if convergence != 0: break
00300

```

```

00301         dataPO[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
00302         dataPO[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit
00303
00304     if show_plot:
00305         plt.plot(dataPO[:,0], dataPO[:,1])
00306         plt.xlabel('Horizontal Displacement [mm]')
00307         plt.ylabel('Horizontal Load [kN]')
00308         plt.title('Pushover curve')
00309         if block:
00310             plt.show()
00311
00312     print("")
00313     print("Pushover complete")
00314     self.load_caseload_case = "Pushover"
00315
00316     wipe()
00317
00318
00319     def LoadingProtocol(self, CtrlNode: int, discr_LP: np.ndarray, timeSeries_ID: int, pattern_ID:
int, Fx = 1*kN_unit, ele_fiber_ID_analysed = -1, fiber_section = 1,
00320     timeSeries_type = "Linear", pattern_type = "Plain", constraints_type = "Plain", numberer_type
= "RCM", system_type = "UmfPack", analysis_type = "Static",
00321     show_plot = True, block = False):
00322         """
00323         Method to perform a loading protocol analysis (displacement-control). If this method is
called, the Pushover and LateralForce methods should be avoided.
00324
00325         @param CtrlNode (int): The node that will be used to impose the displacement from the discr_LP
to perform the analysis.
00326         @param discr_LP (np.ndarray): The loading protocol array (1 dimension) discretised. It needs
to be filled with imposed displacement, not SDR.
00327         Use the functions DiscretizeLoadProtocol and DiscretizeLinearly in FunctionalFeatures
module to help create and/or discretise one.
00328         @param timeSeries_ID (int): ID of the timeseries.
00329         @param pattern_ID (int): ID of the pattern.
00330         @param Fx (float, optional): The force imposed at the control node CtrlNode. It is used for
convergence reasons and it can be arbitrarily small.
00331         Defaults to 1*kN_unit.
00332         @param ele_fiber_ID_analysed (int, optional): The ID of the analysed element with fibers. If
fibers are present in the model and the user wants to save ODB data
00333         (to use in the post-processing with for example FiberResponse), assign to this argument
the ID of the element with fibers chosen.
00334         -1 will ignore the storage of data for fibers. Defaults to -1.
00335         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00336         If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00337         @param timeSeries_type (str, optional): Type of timeseries chosen.
00338         For more information, see the OpenSeesPy documentation. Defaults to "Linear".
00339         @param pattern_type (str, optional): Type of pattern chosen.
00340         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00341         @param constraints_type (str, optional): Type of constraints chosen. It determines how the
constraint equations are enforced in the analysis.
00342         For more information, see the OpenSeesPy documentation. Defaults to "Plain".
00343         @param numberer_type (str, optional): Type of numberer chosen. It determines the mapping
between equation numbers and degrees-of-freedom.
00344         For more information, see the OpenSeesPy documentation. Defaults to "RCM".
00345         @param system_type (str, optional): Type of system of equations chosen. It determines how to
construct the LinearSOE and LinearSolver objects to store and solve the system of equations in the
analysis.
00346         For more information, see the OpenSeesPy documentation. Defaults to "UmfPack".
00347         @param analysis_type (str, optional): Type of analysis chosen. It determines how to construct
the Analysis object, which defines what type of analysis is to be performed.
00348         For more information, see the OpenSeesPy documentation. Defaults to "Static".
00349         @param show_plot (bool, optional): Option to show the 'lateral displacement vs. lateral
loading' curve after the analysis. Defaults to True.
00350         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00351
00352         @exception NegativeValue: The ID of CtrlNode needs to be a positive integer.
00353         @exception NegativeValue: The ID of timeSeries_ID needs to be a positive integer.
00354         @exception NegativeValue: The ID of pattern_ID needs to be a positive integer.
00355         @exception NegativeValue: The ID of fiber_ID_analysed needs to be a positive integer if is
different from -1.
00356         """
00357         if CtrlNode < 1: raise NegativeValue()
00358         if timeSeries_ID < 1: raise NegativeValue()
00359         if pattern_ID < 1: raise NegativeValue()
00360         if ele_fiber_ID_analysed != -1 and ele_fiber_ID_analysed < 1: raise NegativeValue()
00361
00362         # for mass defined: opsplt.createODB(self.name_ODB, "LoadingProtocol", Nmodes = nEigen);
00363         opsplt.createODB(self.name_ODBname_ODB, "LoadingProtocol");
00364         if ele_fiber_ID_analysed != -1: opsplt.saveFiberData2D(self.name_ODBname_ODB,
"LoadingProtocol", ele_fiber_ID_analysed, fiber_section)
00365
00366         # Create load pattern
00367         timeSeries(timeSeries_type, timeSeries_ID)
00368         pattern(pattern_type, timeSeries_ID, pattern_ID)
00369         load(CtrlNode, Fx, 0.0, 0.0) # load(IDNode, Fx, Fy, Mz)

```



```

00370         dU_prev = 0
00371         Nsteps = np.size(discr_LP)          # number of pushover analysis steps
00372
00373         # Set up analysis options
00374         constraints(constraints_type)        # how it handles boundary conditions
00375         numberer(numberer_type)             # renumber dof's to minimize band-width (optimization)
00376         system(system_type)                 # how to store and solve the system of equations in the
analysis
00377                                             # For static model, BandGeneral, for transient and/or big
model, UmfPack
00378         integrator("LoadControl", 1)        # placeholder
00379         algorithm("Newton")                 # placeholder
00380         analysis(analysis_type)             # define type of analysis: static for LoadingProtocol
00381
00382         # Analysis
00383         dataLP = np.zeros((Nsteps+1,2))
00384         next_step = 0
00385         print("")
00386         print("Loading Protocol analysis starts")
00387         for iteration in range(Nsteps):
00388             # Compute displacement using the given loading protocol (discretized)
00389             dU_next = discr_LP[iteration]
00390             dU = dU_next - dU_prev
00391             dU_prev = dU_next
00392
00393             next_step = ProgressingPercentage(Nsteps, iteration, next_step)
00394             convergence = self.__LatDispCtrlLoop__LatDispCtrlLoop(CtrlNode, dU, iteration,
00395                 self.algo, self.test_type, test_type, self.tol, tol, self.test_opt, test_opt,
self.max_iter, max_iter, self.allow_smaller_step, allow_smaller_step)
00396             if convergence != 0: break
00397             dataLP[iteration+1,0] = nodeDisp(CtrlNode, 1)/mm_unit
00398             dataLP[iteration+1,1] = getLoadFactor(pattern_ID)*Fx/kN_unit
00399
00400             if show_plot:
00401                 plt.plot(dataLP[:,0], dataLP[:,1])
00402                 plt.xlabel('Horizontal Displacement [mm]')
00403                 plt.ylabel('Horizontal Load [kN]')
00404                 plt.title('Loading Protocol curve')
00405                 if block:
00406                     plt.show()
00407
00408             print("")
00409             print("Loading Protocol complete")
00410             self.load_case.load_case = "LoadingProtocol"
00411
00412             wipe()
00413
00414
00415         def __LoadCtrlLoop(self, force, iteration: int, algo = "KrylovNewton", test_type = "NormDispIncr",
tol = TOL, test_opt = 0, max_iter = MAX_ITER):
00416             """
00417             PRIVATE METHOD. It is used perform one load increment 'force' load-control analysis step using
'algo' and 'test_type' as algorithm and test.
00418             The integrator is LoadControl. If convergence issues are encountered, the method performs a
convergence analysis trying different ways to converge.
00419
00420             @param force (double): The load increment performed.
00421             @param iteration (int): The current iteration.
00422             @param algo (str, optional): Type of algorithm chosen for the analysis. It determines how to
construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the
non-linear equation.
00423             For more information on the available types, see the OpenSeesPy documentation. Defaults to
"KrylovNewton".
00424             @param test_type (str, optional): Type of test chosen for the analysis. It determines how to
construct a ConvergenceTest object.
00425             Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if
convergence has been achieved at the end of an iteration step.
00426             For more information on the available types, see the OpenSeesPy documentation. Defaults to
"NormDispIncr".
00427             @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to
TOL (from Constants Module).
00428             @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the
iteration
00429             (for example: 0 print nothing and 2 print information on norms and number of iterations at
end of successful test).
00430             For more information, see the OpenSeesPy documentation. Defaults to 0.
00431             @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER
(from Constants Module).
00432
00433             @exception NegativeValue: iteration needs to be a positive integer.
00434             @exception NegativeValue: tol needs to be positive.
00435             @exception NegativeValue: max_iter needs to be positive.
00436
00437             @returns int: 0 if the iteration converged.
00438             """
00439             if iteration < 0: raise NegativeValue()
00440             if tol < 0: raise NegativeValue()

```

```

00441         if max_iter < 0: raise NegativeValue()
00442
00443         # Default analysis
00444         integrator("LoadControl", force)          # LoadControl and DisplacementControl only with
static model, linear TimeSeries w/ factor of 1
00445
00446         test(test_type, tol, max_iter, test_opt)  # Newmark used for transient model
iterations;                                     # type of convergence criteria with tolerance, max
00447
00448         NormDispIncr; optional: test_opt = 2 for debugging  # Normally use EnergyIncr, if conv issues, try
algorithm(algo)                                # use Newton's solution algorithm: updates tangent
00449         stiffness at every iteration
convergence = analyze(1)                        # this will return zero if no convergence problems
00450         were encountered
00451
00452         # Convergence analysis
00453         if convergence != 0:
00454             print("-----")
00455             print("Vertical Load-control analysis failed at iteration {}".format(iteration))
00456             print("Convergence analysis starts")
00457             print("-----")
00458
00459             if convergence != 0:
00460                 print("Try 1")
00461                 convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol, max_iter,
test_opt)
00462
00463             if convergence != 0:
00464                 print("Try 2")
00465                 convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol*10, max_iter*10,
test_opt)
00466
00467             if convergence != 0:
00468                 print("Try 3")
00469                 convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol, max_iter, test_opt)
00470
00471             if convergence != 0:
00472                 print("Try 4")
00473                 convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol*10, max_iter*10,
test_opt)
00474
00475             if convergence != 0:
00476                 print("Try 5")
00477                 convergence = _ConvergenceTest("Newton", "NormDispIncr", tol, max_iter, test_opt)
00478
00479             if convergence != 0:
00480                 print("Try 6")
00481                 convergence = _ConvergenceTest("Newton", "NormDispIncr", tol*10, max_iter*10,
test_opt)
00482
00483             if convergence != 0:
00484                 print("Try 7")
00485                 convergence = _ConvergenceTest("Newton", "EnergyIncr", tol, max_iter, test_opt)
00486
00487             if convergence != 0:
00488                 print("Try 8")
00489                 convergence = _ConvergenceTest("Newton", "EnergyIncr", tol*10, max_iter*10, test_opt)
00490
00491             if convergence != 0:
00492                 print("Try 9")
00493                 convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol, max_iter,
test_opt)
00494
00495             if convergence != 0:
00496                 print("Try 10")
00497                 convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol*10, max_iter*10,
test_opt)
00498
00499             if convergence != 0:
00500                 print("Try 11")
00501                 convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol, max_iter,
test_opt)
00502
00503             if convergence != 0:
00504                 print("Try 12")
00505                 convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol*10, max_iter*10,
test_opt)
00506
00507             if convergence != 0:
00508                 print("")
00509                 print("#####")
00510                 print("NO CONVERGENCE! Load-control analysis stops at iteration {}".format(iteration))
00511                 print("#####")
00512                 print("")

```

```

00512         else:
00513             print("Convergence reached, convergence analysis ends.")
00514
00515         return convergence
00516
00517
00518     def __LatDispCtrlLoop(self, CtrlNode, dU, iteration, algo = "KrylovNewton", test_type =
"NormDispIncr", tol = TOL, test_opt = 0, max_iter = MAX_ITER, allow_smaller_step = False):
00519         """
00520         PRIVATE METHOD. It is used perform one imposed displacement increment 'dU'
displacement-control analysis step using 'algo' and 'test_type' as algorithm and test.
00521         The integrator is DisplacementControl. If convergence issues are encountered, the method
performa a convergence analysis trying different ways to converge.
00522
00523         @param CtrlNode ([type]): [description]
00524         @param dU (float): The imposed displacement increment for the current iteration
00525         @param iteration (int): The current iteration.
00526         @param algo (str, optional): Type of alghoritm chosen for the analysis. It detemines how to
construct a SolutionAlgorithm object, which determines the sequence of steps taken to solve the
non-linear equation.
00527         For more information on the available types, see the OpenSeesPy documentation. Defaults to
"KrylovNewton".
00528         @param test_type (str, optional): Type of test chosen for the analysis. It determines how to
construct a ConvergenceTest object.
00529         Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if
convergence has been achieved at the end of an iteration step.
00530         For more information on the available types, see the OpenSeesPy documentation. Defaults to
"NormDispIncr".
00531         @param tol (float, optional): Tolerance criteria used to check for convergence. Defaults to
TOL (from Constants Module).
00532         @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the
iteration
00533         (for example: 0 print nothing and 2 print information on norms and number of iterations at
end of successful test).
00534         For more information, see the OpenSeesPy documentation. Defaults to 0.
00535         @param max_iter (float, optional): Maximal number of iterations to check. Defaults to MAX_ITER
(from Constants Module).
00536         @param allow_smaller_step (bool, optional): Allow smaller steps in the displacement-control
analysis. Defaults to False.
00537
00538         @exception NegativeValue: iteration needs to be a positive integer.
00539         @exception NegativeValue: tol needs to be positive.
00540         @exception NegativeValue: max_iter needs to be positive.
00541
00542         @returns int: 0 if the interation converged.
00543         """
00544         if iteration < 0: raise NegativeValue()
00545         if tol < 0: raise NegativeValue()
00546         if max_iter < 0: raise NegativeValue()
00547
00548         # Default analysis
00549         CtrlDOF = 1
00550         integrator("DisplacementControl", CtrlNode, CtrlDOF, dU, 1, dU, dU) # use
displacement-controlled analysis
00551         test(test_type, tol, max_iter, test_opt) # type of convergence criteria with tolerance, max
iterations;
00552         NormDispIncr; optional: test_opt = 2 for debugging # Normally use EnergyIncr, if conv issues, try
algorithm(algo) # use Newton's solution algorithm: updates tangent
00553         stiffness at every iteration
00554         convergence = analyze(1) # this will return zero if no convergence problems
were encountered
00555
00556         # Convergence analysis
00557         if convergence != 0:
00558
00559             print("-----")
00560             print("Lateral Displacement-control analysis failed at iteration {} and control node
lateral displacement = {} mm.".format(iteration, nodeDisp(CtrlNode, CtrlDOF)/mm_unit))
00561             print("Convergence analysis starts")
00562             print("-----")
00563
00564             if convergence != 0:
00565                 print("Try 1")
00566                 convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol, max_iter,
test_opt)
00567
00568             if convergence != 0:
00569                 print("Try 2")
00570                 convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol*10, max_iter*10,
test_opt)
00571
00572             if convergence != 0:
00573                 print("Try 3")
00574                 convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol, max_iter, test_opt)

```

```

00575         if convergence != 0:
00576             print("Try 4")
00577             convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol*10, max_iter*10,
test_opt)
00578
00579         if convergence != 0:
00580             print("Try 5")
00581             convergence = _ConvergenceTest("Newton", "NormDispIncr", tol, max_iter, test_opt)
00582
00583         if convergence != 0:
00584             print("Try 6")
00585             convergence = _ConvergenceTest("Newton", "NormDispIncr", tol*10, max_iter*10,
test_opt)
00586
00587         if convergence != 0:
00588             print("Try 7")
00589             convergence = _ConvergenceTest("Newton", "EnergyIncr", tol, max_iter, test_opt)
00590
00591         if convergence != 0:
00592             print("Try 8")
00593             convergence = _ConvergenceTest("Newton", "EnergyIncr", tol*10, max_iter*10, test_opt)
00594
00595         if convergence != 0:
00596             print("Try 9")
00597             convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol, max_iter,
test_opt)
00598
00599         if convergence != 0:
00600             print("Try 10")
00601             convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol*10, max_iter*10,
test_opt)
00602
00603         if convergence != 0:
00604             print("Try 11")
00605             convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol, max_iter,
test_opt)
00606
00607         if convergence != 0:
00608             print("Try 12")
00609             convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol*10, max_iter*10,
test_opt)
00610
00611         if convergence != 0 and allow_smaller_step:
00612             print("Use smaller steps")
00613             print("10 times more intergator iteration, min_dU = dU/10")
00614             integrator("DisplacementControl", CtrlNode, CtrlDOF, dU, 10, dU/10, dU)
00615
00616         if convergence != 0:
00617             print("Try 13")
00618             convergence = _ConvergenceTest("KrylovNewton", "NormDispIncr", tol, max_iter,
test_opt)
00619
00620         if convergence != 0:
00621             print("Try 14")
00622             convergence = _ConvergenceTest("KrylovNewton", "EnergyIncr", tol, max_iter,
test_opt)
00623
00624         if convergence != 0:
00625             print("Try 15")
00626             convergence = _ConvergenceTest("Newton", "NormDispIncr", tol, max_iter, test_opt)
00627
00628         if convergence != 0:
00629             print("Try 16")
00630             convergence = _ConvergenceTest("Newton", "EnergyIncr", tol, max_iter, test_opt)
00631
00632         if convergence != 0:
00633             print("Try 17")
00634             convergence = _ConvergenceTest("ModifiedNewton", "NormDispIncr", tol, max_iter,
test_opt)
00635
00636         if convergence != 0:
00637             print("Try 19")
00638             convergence = _ConvergenceTest("ModifiedNewton", "EnergyIncr", tol, max_iter,
test_opt)
00639
00640         if convergence != 0:
00641             print("")
00642             print("#####")
00643             print("NO CONVERGENCE! Lateral Displacement-control analysis stops at iteration {} and
control node lateral displacement = {} mm.".format(iteration, nodeDisp(CtrlNode, CtrlDOF)/mm_unit))
00644             print("#####")
00645             print("")
00646         else:
00647             print("Convergence reached, convergence analysis ends.")
00648
00649         return convergence
00650

```

```

00651
00652     def DeformedShape(self, scale = 1, animate = False, dt = 0.01):
00653         """
00654         Method that shows the final deformed shape of the model. It can also show the animation that
00655         shows how the model behaved during the analysis.
00656
00657         @param scale (int, optional): The scaling factor to magnify the deformation. The value should
00658         be adjusted for each model. Defaults to 1.
00659         @param animate (bool, optional): Option to show the animation of the model during the
00660         analysis. Defaults to False.
00661         @param dt (float, optional): The time step between every iteration. Defaults to 0.01.
00662
00663         @exception NameError: The methods for the analysis were not called.
00664         """
00665         if self.load_caseload_case == "None": raise NameError("The analysis is not complete.")
00666
00667         # Display deformed shape, the scaling factor needs to be adjusted for each model
00668         opsplt.plot_deformedshape(Model = self.name_ODBname_ODB, LoadCase=self.load_caseload_case,
00669         scale = scale)
00670         if animate:
00671             opsplt.animate_deformedshape(Model = self.name_ODBname_ODB,
00672             LoadCase=self.load_caseload_case, dt = dt, scale = scale)
00673
00674     def FiberResponse(self, ele_fiber_ID_analysed: int, fiber_section = 1, animate_stress = False,
00675     animate_strain = False, fps = 25):
00676         """
00677         Method that shows the final stress response of the fiber section chosen.
00678         It can also show the animation that shows how the fiber section behaved during the analysis.
00679         The fiber ID and section needs to be recorded during the analysis,
00680         thus if the method LateralForce, Pushover or LoadingProtocol was used, the same fiber ID and
00681         section need to be used.
00682
00683         @param ele_fiber_ID_analysed (int): The ID of the analysed fiber. If fibers are present in the
00684         model and the user wants to save ODB data
00685         (to use in the post-processing with for example FiberResponse), assign to this argument
00686         the ID of the fiber chosen.
00687         -1 will ignore the storage of data for fibers.
00688         @param fiber_section (int, optional): The section number, i.e. the Gauss integratio number.
00689         If the fiber_ID_analysed is equal to -1, this argument is not used. Defaults to 1.
00690         @param animate_stress (bool, optional): Option to show the animation of the fiber stress
00691         during the analysis. Defaults to False.
00692         @param animate_strain (bool, optional): Option to show the animation of the fiber strain
00693         during the analysis. Defaults to False.
00694         @param fps (int, optional): Number of frame per seconds for the animations. Defaults to 25.
00695         @exception NameError: The methods for the analysis were not called.
00696         """
00697         if self.load_caseload_case == "None": raise NameError("The analysis is not complete.")
00698
00699         opsplt.plot_fiberResponse2D(self.name_ODBname_ODB, self.load_caseload_case,
00700         ele_fiber_ID_analysed, fiber_section, InputType = 'stress')
00701         if animate_stress:
00702             anil = opsplt.animate_fiberResponse2D(self.name_ODBname_ODB, self.load_caseload_case,
00703             ele_fiber_ID_analysed, fiber_section, InputType = 'stress', fps = fps)
00704         if animate_strain:
00705             anil = opsplt.animate_fiberResponse2D(self.name_ODBname_ODB, self.load_caseload_case,
00706             ele_fiber_ID_analysed, fiber_section, InputType = 'strain', fps = fps)
00707
00708 def _ConvergenceTest(algo_type: str, test_type: str, tol, max_iter, test_opt = 0):
00709     """
00710     PRIVATE FUNCTION. It is used during the convergence analysis to test different ways to reach
00711     convergence.
00712
00713     @param algo_type (str): Type of algorithmt chosen for the analysis. It detemines how to construct a
00714     SolutionAlgorithm object, which determines the sequence of steps taken to solve the non-linear
00715     equation.
00716     For more information on the available types, see the OpenSeesPy documentation.
00717     @param test_type (str): Type of test chosen for the analysis. It determines how to construct a
00718     ConvergenceTest object.
00719     Certain SolutionAlgorithm objects require a ConvergenceTest object to determine if convergence
00720     has been achieved at the end of an iteration step.
00721     For more information on the available types, see the OpenSeesPy documentation.
00722     @param tol (float): Tolerance criteria used to check for convergence.
00723     @param max_iter (float): Maximal number of iterations to check.
00724     @param test_opt (int, optional): Print-flag from 0 to 5 used to receive more info during the
00725     iteration
00726     (for example: 0 print nothing and 2 print information on norms and number of iterations at end
00727     of successful test).
00728     For more information, see the OpenSeesPy documentation. Defaults to 0.
00729
00730     @returns int: 0 if the interation converged.
00731     """
00732     print("algorithm: {}".format(algo_type))
00733     print("test: {}, tol = {}, max iter = {}".format(test_type, tol, max_iter))
00734     test(test_type, tol, max_iter, test_opt)
00735     algorithm(algo_type)

```

```

00716
00717     return analyze(1)
00718
00719
00720

```

## 8.3 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/↵ Connections.py File Reference

### Namespaces

- namespace [Connections](#)

*Module with different functions useful when defining boundary conditions (fix) or connections (pin, rigid or springs).*

### Functions

- def [Pin](#) (int NodeRID, int NodeCID)  
*Function that constrains the translational DOF with a multi-point constraint.*
- def [RigidSupport](#) (int NodeID)  
*Function that fixes the x, y movements and the rotation of one node.*
- def [RotationalSpring](#) (int ElementID, int NodeRID, int NodeCID, int MatID, Rigid=False)  
*Function that defines a zero-length spring and constrains the translations DOFs of the spring.*

## 8.4 Connections.py

[Go to the documentation of this file.](#)

```

00001 """Module with different functions useful when defining boundary conditions (fix) or connections (pin,
00002     rigid or springs). \n
00003     Carmine Schipani, 2021
00004 """
00005 from openseespy.opensees import *
00006 from OpenSeesPyAssistant.ErrorHandling import *
00007
00008
00009 def RigidSupport(NodeID: int):
00010     """
00011     Function that fixes the x, y movements and the rotation of one node.
00012
00013     @param NodeID (int): ID of the node to be fixed
00014
00015     @exception NegativeValue: The ID of NodeID needs to be a positive integer.
00016     """
00017     if NodeID < 1: raise NegativeValue()
00018
00019     fix(NodeID, 1, 1, 1)
00020
00021
00022 def Pin(NodeRID: int, NodeCID: int):
00023     """
00024     Function that constrains the translational DOF with a multi-point constraint.
00025
00026     @param NodeRID (int): Node ID which will be retained by the multi-point constraint
00027     @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
00028
00029     @exception WrongArgument: The IDs passed needs to be different.
00030     @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
00031     @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
00032     """
00033     if NodeCID == NodeRID: raise WrongArgument()
00034     if NodeRID < 1: raise NegativeValue()
00035     if NodeCID < 1: raise NegativeValue()
00036
00037     # Constrain the translational DOF with a multi-point constraint

```

```

00038     #           retained constrained DOF_1 DOF_2
00039     equalDOF(NodeRID, NodeCID, 1, 2)
00040
00041
00042 def RotationalSpring(ElementID: int, NodeRID: int, NodeCID: int, MatID: int, Rigid = False):
00043     """
00044     Function that defines a zero-length spring and constrains the translations DOFs of the spring. Can
00045     be used also to create rigid connections.
00046
00047     @param ElementID (int): ID of the zerolength element that models the spring
00048     @param NodeRID (int): Node ID which will be retained by the multi-point constraint
00049     @param NodeCID (int): Node ID which will be constrained by the multi-point constraint
00050     @param MatID (int): ID of the material model chosen
00051     @param Rigid (bool, optional): Optional argument that transforms the joint in a completely rigid
00052     connection. Defaults to False.
00053
00054     @exception NegativeValue: The ID of ElementID needs to be a positive integer.
00055     @exception NegativeValue: The ID of NodeCID needs to be a positive integer.
00056     @exception NegativeValue: The ID of NodeRID needs to be a positive integer.
00057     @exception WrongArgument: The IDs of the nodes passed needs to be different.
00058     @exception NegativeValue: The ID of MatID needs to be a positive integer.
00059
00060     """
00061     if ElementID < 1: raise NegativeValue()
00062     if NodeCID < 1: raise NegativeValue()
00063     if NodeRID < 1: raise NegativeValue()
00064     if NodeCID == NodeRID: raise WrongArgument()
00065     if MatID < 1: raise NegativeValue()
00066
00067     if not Rigid:
00068         # Zero length element (spring)
00069         element("zeroLength", ElementID, NodeRID, NodeCID, "-mat", MatID, "-dir", 6)
00070
00071         # Constrain the translational DOF with a multi-point constraint
00072         Pin(NodeRID, NodeCID)
00073     else:
00074         equalDOF(NodeRID, NodeCID, 1, 2, 3)
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000

```

## 8.5 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/↵ Constants.py File Reference

### Namespaces

- namespace [Constants](#)  
*Module with the values of a set of essential constants.*

### Variables

- float [G\\_CONST](#) = 9.810\*m\_unit/s\_unit\*\*2
- int [MAX\\_ITER](#) = 100
- int [MAX\\_ITER\\_INTEGRATION](#) = 50
- float [RIGID](#) = 100.0
- float [TOL](#) = 1.0e-6
- float [TOL\\_INTEGRATION](#) = 1.0e-12
- float [ZERO](#) = 1.0e-9

## 8.6 Constants.py

[Go to the documentation of this file.](#)

```
00001 """Module with the values of a set of essential constants. They are consistent with the units defined
      in Units. \n
00002 Carmine Schipani, 2021
00003 """
00004
00005 from OpenSeesPyAssistant.Units import *
00006
00007
00008 TOL = 1.0e-6
00009 TOL_INTEGRATION = 1.0e-12
00010 ZERO = 1.0e-9          # used when defining mass that is equal to 0 (avoid convergence
                        problem)
00011 G_CONST = 9.810*m_unit/s_unit**2
00012 RIGID = 100.0          # multiply with a mechanical or geometrical value to have the
                        corresponding infinitely rigid value
00013 MAX_ITER = 100
00014 MAX_ITER_INTEGRATION = 50
```

## 8.7 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Data↵ Management.py File Reference

### Classes

- class [DataManagement](#)  
*Abstract parent class for data management.*

### Namespaces

- namespace [DataManagement](#)  
*Module with the parent abstract class DataManagement.*

## 8.8 DataManagement.py

[Go to the documentation of this file.](#)

```
00001 """
00002 Module with the parent abstract class DataManagement. \n
00003 Carmine Schipani, 2021
00004 """
00005
00006 from abc import ABC, abstractmethod
00007 from OpenSeesPyAssistant.ErrorHandling import *
00008 import numpy as np
00009
00010
00011 class DataManagement(ABC):
00012     """
00013     Abstract parent class for data management.
00014     Using the associated MATLAB class \n
00015     LOAD_CLASS.m \n
00016     for the postprocessing in MATLAB, allowing for simpler and more reliable data management because
00017     the parameters
00018     from the OpenSeesPy analysis are imported automatically.
00019     """
00020     def SaveData(self, f):
00021         """
00022         Function that lists in the command window and saves in a opened file text "f" the data from
00023         the "self" class that calls it.
00024         Example: call this function after this line: \n
00025         with open(FileName, 'w') as f:
00026             @param f (io.TextIOWrapper): Opened file to write into
```



```

00027
00028     @exception WrongDimension: The number of lists in the list self.data needs to be 2
00029     """
00030     if len(self.data[0]) != 2: raise WrongDimension()
00031
00032     delimiter = "##### " # 30 times #
00033     col_delimiter = "\t" # tab
00034     for data_line in self.data:
00035         f.write('\n')
00036         for col in data_line:
00037             if type(col) == np.ndarray:
00038                 tmp_str = np.array_str(col, max_line_width = np.inf)
00039             else:
00040                 tmp_str = str(col)
00041             f.write(tmp_str)
00042             f.write(col_delimiter)
00043         f.write('\n')
00044         f.write('NEW INFO SECTION DELIMITER \t')
00045         f.write(delimiter)
00046
00047     @abstractmethod
00048     def ShowInfo(self):
00049         """
00050         Abstract method that shows the data stored in the class in the command window.
00051         In some cases, it's possible to plot some information (for example the curve of the material
00052         model).
00053         """
00054         pass
00055
00056     @abstractmethod
00057     def ReInit(self):
00058         """
00059         Abstract method that computes the value of the parameters with respect of the arguments. \n
00060         Use after changing the value of argument inside the class (to update the values accordingly).
00061         \n
00062         This function can be very useful in combination with the function "deepcopy()" from the module
00063         "copy". \n
00064         Be careful that the parameter self.Initialized is also copied, thus it is safer to copy the
00065         class before the method that calls the actual OpenSees commands (and initialise the object).
00066         """
00067         pass
00068
00069     @abstractmethod
00070     def UpdateStoredData(self):
00071         """
00072         Abstract method used to define and update the self.data member variable. \n
00073         This member variable (self.data) is a list of lists with 2 entries (info_name and info_value)
00074         and for each list is stored a different member variable of the class. \n
00075         Useful to debug the model, export data, copy object.
00076         """
00077         pass

```

## 8.9 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandling.py File Reference ↩

### Classes

- class [InconsistentGeometry](#)  
*Exception class for the "inconsistent geometry" error.*
- class [MemberFailure](#)  
*Exception class for the "member failure" error.*
- class [NegativeValue](#)  
*Exception class for the "negative value (argument or result)" error.*
- class [NoApplicability](#)  
*Exception class for the "no applicability of formula of theory" error.*
- class [PositiveValue](#)  
*Exception class for the "positive value (argument or result)" error.*
- class [WrongArgument](#)  
*Exception class for the "input of a wrong argument" error.*
- class [WrongDimension](#)

*Exception class for the "wrong array dimensions" error.*

- class [WrongNodeIDConvention](#)

*Exception class for the "wrong node ID convention definition" error.*

- class [ZeroDivision](#)

*Exception class for the "zero division" error.*

- class [ZeroLength](#)

*Exception class for the "zero length element (non intentional)" error.*

## Namespaces

- namespace [ErrorHandling](#)

*Module dedicated to the error handling.*

## 8.10 ErrorHandling.py

[Go to the documentation of this file.](#)

```
00001 """Module dedicated to the error handling. \n
00002 Carmine Schipani, 2021
00003 """
00004
00005
00006 class ZeroDivision(Exception):
00007     """Exception class for the "zero division" error.
00008     """
00009     pass
00010
00011 class WrongArgument(Exception):
00012     """Exception class for the "input of a wrong argument" error.
00013     """
00014     pass
00015
00016 class NegativeValue(Exception):
00017     """Exception class for the "negative value (argument or result)" error.
00018     """
00019     pass
00020
00021 class PositiveValue(Exception):
00022     """Exception class for the "positive value (argument or result)" error.
00023     """
00024     pass
00025
00026 class WrongDimension(Exception):
00027     """Exception class for the "wrong array dimensions" error.
00028     """
00029     pass
00030
00031 class InconsistentGeometry(Exception):
00032     """Exception class for the "inconsistent geometry" error.
00033     """
00034     pass
00035
00036 class MemberFailure(Exception):
00037     """Exception class for the "member failure" error.
00038     """
00039     pass
00040
00041 class WrongNodeIDConvention(Exception):
00042     """Exception class for the "wrong node ID convention definition" error.
00043     """
00044     def __init__(self, node):
00045         self.nodenode = node
00046
00047 class NoApplicability(Exception):
00048     """Exception class for the "no applicability of formula of theory" error.
00049     """
00050     pass
00051
00052 class ZeroLength(Exception):
00053     """Exception class for the "zero length element (non intentional)" error.
00054     """
00055     def __init__(self, element):
00056         self.elementelement = element
```

## 8.11 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers.py File Reference

### Classes

- class [Fibers](#)  
*Parent abstract class for the storage and manipulation of a fiber's information (mechanical and geometrical parameters, etc) and initialisation in the model.*
- class [FibersCirc](#)  
*Class that stores functions, material properties, geometric and mechanical parameters for a circular RC fiber section.*
- class [FibersCircRCCircShape](#)  
*Class that is the children of [FibersCirc](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.*
- class [FibersIShape](#)  
*Class that stores functions, material properties, geometric and mechanical parameters for a steel I shape (non double symmetric) fiber section.*
- class [FibersIShapeSteelIShape](#)  
*Class that is the children of [FibersIShape](#) and combine the class [SteelIShape](#) (section) to retrieve the information needed.*
- class [FibersRect](#)  
*Class that stores functions, material properties, geometric and mechanical parameters for a rectangular RC fiber section.*
- class [FibersRectRCRectShape](#)  
*Class that is the children of [FibersRect](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.*

### Namespaces

- namespace [Fibers](#)  
*Module for the fibers (rectangular, circular and I shape).*

### Functions

- def [create\\_fiber\\_section](#) (fiber\_info)  
*Initialise fiber cross-section with OpenSeesPy commands.*
- def [plot\\_fiber\\_section](#) (fiber\_info, fill\_shapes=True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g', 'y'])  
*Plot fiber cross-section.*

## 8.12 Fibers.py

[Go to the documentation of this file.](#)

```
00001 """
00002 Module for the fibers (rectangular, circular and I shape).
00003 Carmine Schipani, 2021
00004 """
00005
00006 from openseespy.opensees import *
00007 import matplotlib.pyplot as plt
00008 from matplotlib.patches import Circle, Polygon, Wedge
00009 import numpy as np
00010 from copy import copy, deepcopy
00011 from OpenSeesPyAssistant.Section import *
00012 from OpenSeesPyAssistant.DataManagement import *
```

```

00013 from OpenSeesPyAssistant.ErrorHandling import *
00014 from OpenSeesPyAssistant.Units import *
00015 from OpenSeesPyAssistant.MaterialModels import *
00016
00017
00018 class Fibers(DataManagement):
00019     """
00020     Parent abstract class for the storage and manipulation of a fiber's information (mechanical
00021     and geometrical parameters, etc) and initialisation in the model.
00022
00023     @param DataManagement: Parent abstract class.
00024     """
00025     pass
00026
00027
00028 class FibersRect(Fibers):
00029     """
00030     Class that stores functions, material properties, geometric and mechanical parameters for a
00031     rectangular RC fiber section.
00032     Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more
00033     information, see the OpenSeesPy documentation.
00034
00035     @param Fibers: Parent abstract class.
00036     """
00037     def __init__(self, ID: int, b, d, Ay, D_hoops, e, unconf_mat_ID: int, conf_mat_ID: int,
00038                  bars_mat_ID: int,
00039                  bars_x: np.ndarray, ranges_y: np.ndarray, discr_core: list, discr_cover_lateral: list,
00040                  discr_cover_topbottom: list, GJ = 0.0):
00041         """
00042         Constructor of the class.
00043
00044         @param ID (int): Unique fiber section ID.
00045         @param b (float): Width of the section.
00046         @param d (float): Depth of the section.
00047         @param Ay (float): Area of one vertical reinforcing bar.
00048         @param D_hoops (float): Diameter of the hoops.
00049         @param e (float): Concrete cover.
00050         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
00051         fibers.
00052         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00053         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
00054         fibers.
00055         @param bars_x (np.ndarray): Array with a range of aligned vertical reinforcing bars for each
00056         row in x direction.
00057         Distances from border to bar centerline, bar to bar centerlines and finally bar centerline
00058         to border in the x direction (aligned).
00059         Starting from the left to right, from the top range to the bottom one.
00060         The number of bars for each range can vary; in this case, add this argument when defining
00061         the array " dtype = object"
00062         @param ranges_y (np.ndarray): Array of dimension 1 with the position or spacing in y of the
00063         ranges in bars_x.
00064         Distances from border to range centerlines, range to range centerlines and finally range
00065         centerline to border in the y direction.
00066         Starting from the top range to the bottom one.
00067         @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
00068         confined core.
00069         @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK
00070         (y) for the lateral unconfined cover.
00071         @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK
00072         (y) for the top and bottom unconfined cover.
00073         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
00074         Defaults to 0.0, assume no torsional stiffness.
00075
00076         @exception NegativeValue: ID needs to be a positive integer.
00077         @exception NegativeValue: b needs to be positive.
00078         @exception NegativeValue: d needs to be positive.
00079         @exception NegativeValue: Ay needs to be positive.
00080         @exception NegativeValue: D_hoops needs to be positive.
00081         @exception NegativeValue: e needs to be positive.
00082         @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
00083         @exception NegativeValue: conf_mat_ID needs to be a positive integer.
00084         @exception NegativeValue: bars_mat_ID needs to be a positive integer.
00085         @exception WrongDimension: Number of rows in the list bars_x needs to be the same of the
00086         length of ranges_y - 1.
00087         @exception InconsistentGeometry: The sum of the distances for each row in bars_x should be
00088         equal to the section's width (tol = 5 mm).
00089         @exception InconsistentGeometry: The sum of the distances in ranges_y should be equal to the
00090         section's depth (tol = 5 mm).
00091         @exception InconsistentGeometry: e should be smaller than half the depth and the width of the
00092         section.
00093         @exception WrongDimension: discr_core has a length of 2.
00094         @exception WrongDimension: discr_cover_lateral has a length of 2.
00095         @exception WrongDimension: discr_cover_topbottom has a length of 2.
00096         @exception NegativeValue: GJ needs to be positive.
00097         """
00098         # Check
00099         if ID < 1: raise NegativeValue()

```

```

00081         if b < 0: raise NegativeValue()
00082         if d < 0: raise NegativeValue()
00083         if Ay < 0: raise NegativeValue()
00084         if D_hoops < 0: raise NegativeValue()
00085         if e < 0: raise NegativeValue()
00086         if unconf_mat_ID < 1: raise NegativeValue()
00087         if conf_mat_ID < 1: raise NegativeValue()
00088         if bars_mat_ID < 1: raise NegativeValue()
00089         if np.size(bars_x) != np.size(ranges_y)-1: raise WrongDimension()
00090         geometry_tol = 5*mm_unit
00091         for bars in bars_x:
00092             if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
00093             if abs(np.sum(ranges_y) - d) > geometry_tol: raise InconsistentGeometry()
00094             if e > b/2 or e > d/2: raise InconsistentGeometry()
00095             if len(discr_core) != 2: raise WrongDimension()
00096             if len(discr_cover_lateral) != 2: raise WrongDimension()
00097             if len(discr_cover_topbottom) != 2: raise WrongDimension()
00098             if GJ < 0: raise NegativeValue()
00099
00100         # Arguments
00101         self.IDID = ID
00102         self.bb = b
00103         self.dd = d
00104         self.AyAy = Ay
00105         self.D_hoopsD_hoops = D_hoops
00106         self.ee = e
00107         self.unconf_mat_IDunconf_mat_ID = unconf_mat_ID
00108         self.conf_mat_IDconf_mat_ID = conf_mat_ID
00109         self.bars_mat_IDbars_mat_ID = bars_mat_ID
00110         self.bars_xbars_x = deepcopy(bars_x)
00111         self.ranges_yranges_y = copy(ranges_y)
00112         self.discr_corediscr_core = copy(discr_core)
00113         self.discr_cover_lateraldiscr_cover_lateral = copy(discr_cover_lateral)
00114         self.discr_cover_topbottomdiscr_cover_topbottom = copy(discr_cover_topbottom)
00115         self.GJGJ = GJ
00116
00117         # Initialized the parameters that are dependent from others
00118         self.section_name_tagsection_name_tag = "None"
00119         self.InitializedInitialized = False
00120         self.ReInitReInit()
00121
00122     def ReInit(self):
00123         """
00124         Implementation of the homonym abstract method.
00125         See parent class DataManagement for detailed information.
00126         """
00127         # Memebers
00128         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
00129
00130         # Parameters
00131         z1 = self.bb/2
00132         y1 = self.dd/2
00133         zc = z1-self.ee-self.D_hoopsD_hoops/2
00134         yc = y1-self.ee-self.D_hoopsD_hoops/2
00135
00136         # Create the concrete core fibers
00137         core = [-yc, -zc, yc, zc]
00138         core_cmd = ['patch', 'rect', self.conf_mat_IDconf_mat_ID, *self.discr_corediscr_core, *core]
00139
00140         # Create the concrete cover fibers (bottom left, top right)
00141         cover_up = [yc, -z1, y1, z1]
00142         cover_down = [-y1, -z1, -yc, z1]
00143         cover_left = [-yc, zc, yc, z1]
00144         cover_right = [-yc, -z1, yc, -zc]
00145         cover_up_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
*self.discr_cover_topbottomdiscr_cover_topbottom, *cover_up]
00146         cover_down_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
*self.discr_cover_topbottomdiscr_cover_topbottom, *cover_down]
00147         cover_left_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
*self.discr_cover_lateraldiscr_cover_lateral, *cover_left]
00148         cover_right_cmd = ['patch', 'rect', self.unconf_mat_IDunconf_mat_ID,
*self.discr_cover_lateraldiscr_cover_lateral, *cover_right]
00149         self.fib_secfib_sec = [['section', 'Fiber', self.IDID, '-GJ', self.GJGJ],
core_cmd, cover_up_cmd, cover_down_cmd, cover_left_cmd, cover_right_cmd]
00150
00151         # Create the reinforcing fibers (top, middle, bottom)
00152         # NB: note the order of definition of bars_x and ranges_y
00153         nr_bars = 0
00154         for range in self.bars_xbars_x:
00155             nr_bars += np.size(range)-1
00156         rebarY = -np.cumsum(self.ranges_yranges_y[0:-1]) + y1
00157         self.rebarYZrebarYZ = np.zeros((nr_bars, 2))
00158
00159         iter = 0
00160         for ii, Y in enumerate(rebarY):
00161             rebarZ = -np.cumsum(self.bars_xbars_x[ii][0:-1]) + z1

```

```

00163         for Z in rebarZ:
00164             self.rebarYZrebarYZ[iter, :] = [Y, Z]
00165             iter = iter + 1
00166
00167     for YZ in self.rebarYZrebarYZ:
00168         self.fib_secfib_sec.append(['layer', 'bar', self.bars_mat_IDbars_mat_ID, self.AyAy, *YZ])
00169
00170     # Data storage for loading/saving
00171     self.UpdateStoredDataUpdateStoredData()
00172
00173
00174     # Methods
00175     def UpdateStoredData(self):
00176         """
00177         Implementation of the homonym abstract method.
00178         See parent class DataManagement for detailed information.
00179         """
00180         self.data = [{"INFO_TYPE": "FibersRect", # Tag for differentiating different data
00181                      ["ID", self.IDID],
00182                      ["section_name_tag", self.section_name_tagsection_name_tag],
00183                      ["b", self.bb],
00184                      ["d", self.dd],
00185                      ["Ay", self.AyAy],
00186                      ["D_hoops", self.D_hoopsD_hoops],
00187                      ["e", self.ee],
00188                      ["GJ", self.GJGJ],
00189                      ["conf_mat_ID", self.conf_mat_IDconf_mat_ID],
00190                      ["discr_core", self.discr_corediscr_core],
00191                      ["unconf_mat_ID", self.unconf_mat_IDunconf_mat_ID],
00192                      ["discr_cover_topbottom", self.discr_cover_topbottomdiscr_cover_topbottom],
00193                      ["discr_cover_lateral", self.discr_cover_lateraldiscr_cover_lateral],
00194                      ["bars_mat_ID", self.bars_mat_IDbars_mat_ID],
00195                      ["bars_x", self.bars_xbars_x],
00196                      ["ranges_y", self.ranges_yranges_y],
00197                      ["Initialized", self.InitializedInitialized]]
00198
00199     def ShowInfo(self, plot = False, block = False):
00200         """
00201         Implementation of the homonym abstract method.
00202         See parent class DataManagement for detailed information.
00203
00204         @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
00205         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00206         """
00207         print("")
00208         print("Requested info for FibersRect, ID = {}".format(self.IDID))
00209         print("Section associated: {}".format(self.section_name_tagsection_name_tag))
00210         print("Base b = {} mm and depth d = {} mm".format(self.bb/mm_unit, self.dd/mm_unit))
00211         print("Confined material model ID = {}".format(self.conf_mat_IDconf_mat_ID))
00212         print("Unconfined material model ID = {}".format(self.unconf_mat_IDunconf_mat_ID))
00213         print("Bars material model ID = {}".format(self.bars_mat_IDbars_mat_ID))
00214         print("Discretisation in the core [IJ or x/z dir, JK or y dir] = {}".format(self.discr_corediscr_core))
00215         print("Discretisation in the lateral covers [IJ or x/z dir, JK or y dir] = {}".format(self.discr_cover_lateraldiscr_cover_lateral))
00216         print("Discretisation in the top and bottom covers [IJ or x/z dir, JK or y dir] = {}".format(self.discr_cover_topbottomdiscr_cover_topbottom))
00217         print("")
00218
00219         if plot:
00220             plot_fiber_section(self.fib_secfib_sec, matcolor=['#808080', '#D3D3D3', 'k'])
00221
00222             if block:
00223                 plt.show()
00224
00225
00226     def CreateFibers(self):
00227         """
00228         Method that initialises the fiber by calling the OpenSeesPy commands.
00229         """
00230         create_fiber_section(self.fib_secfib_sec)
00231         self.InitializedInitialized = True
00232         self.UpdateStoredDataUpdateStoredData()
00233
00234
00235     class FibersRectRRectShape(FibersRect):
00236         """
00237         Class that is the children of FibersRect and combine the class RRectShape (section) to retrieve
the information needed.
00238
00239         @param FibersRect: Parent class.
00240         """
00241         def __init__(self, ID: int, section: RRectShape, unconf_mat_ID: int, conf_mat_ID: int,
bars_mat_ID: int,
discr_core: list, discr_cover_lateral: list, discr_cover_topbottom: list, GJ=0):
00242             """
00243

```

```

00244         Constructor of the class.
00245
00246         @param ID (int): Unique fiber section ID.
00247         @param section (RCRectShape): RCRectShape section object.
00248         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
fibers.
00249         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00250         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
fibers.
00251         @param discr_core (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
confined core.
00252         @param discr_cover_lateral (list): List with two entries: discretisation in IJ (x/z) and JK
(y) for the lateral unconfined core.
00253         @param discr_cover_topbottom (list): List with two entries: discretisation in IJ (x/z) and JK
(y) for the top and bottom unconfined core.
00254         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00255         """
00256         self.sectionsection = deepcopy(section)
00257         super().__init__(ID, section.b, section.d, section.Ay, section.D_hoops, section.e,
unconf_mat_ID, conf_mat_ID, bars_mat_ID,
00258             section.bars_position_x, section.bars_ranges_position_y, discr_core, discr_cover_lateral,
discr_cover_topbottom, GJ=GJ)
00259         self.section_name_tagsection_name_tag = section.name_tag
00260         self.UpdateStoredDataUpdateStoredData()
00261
00262
00263 class FibersCirc(Fibers):
00264     """
00265     Class that stores functions, material properties, geometric and mechanical parameters for a
circular RC fiber section.
00266     Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more
information, see the OpenSeesPy documentation.
00267
00268     @param Fibers: Parent abstract class.
00269     """
00270     def __init__(self, ID: int, b, e, D_bars, Ay, n_bars, D_hoops, unconf_mat_ID: int, conf_mat_ID:
int, bars_mat_ID: int,
00271         discr_core: list, discr_cover: list, alpha_i = 0.0, GJ = 0.0):
00272         """
00273         Constructor of the class.
00274
00275         @param ID (int): Unique fiber section ID.
00276         @param b (float): Width of the section.
00277         @param e (float): Concrete cover.
00278         @param D_bars (float): Diameter of vertical reinforcing bars.
00279         @param Ay (float): Area of one vertical reinforcing bar.
00280         @param n_bars (float): Number of reinforcement (allow float for computing the equivalent
n_bars with different reinforcement areas).
00281         @param D_hoops (float): Diameter of the hoops.
00282         @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
fibers.
00283         @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00284         @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
fibers.
00285         @param discr_core (list): List with two entries: number of subdivisions (fibers) in the
circumferential direction (number of wedges),
00286             number of subdivisions (fibers) in the radial direction (number of rings) for the confined
core.
00287         @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the
circumferential direction (number of wedges),
00288             number of subdivisions (fibers) in the radial direction (number of rings) for the
unconfined cover.
00289         @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to
the y axis, counterclockwise. Defaults to 0.0.
00290         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00291
00292         @exception NegativeValue: ID needs to be a positive integer.
00293         @exception NegativeValue: b needs to be positive.
00294         @exception NegativeValue: e needs to be positive.
00295         @exception InconsistentGeometry: e can't be bigger than half of the width b.
00296         @exception NegativeValue: D_bars needs to be positive.
00297         @exception NegativeValue: Ay needs to be positive.
00298         @exception NegativeValue: n_bars needs to be positive.
00299         @exception NegativeValue: D_hoops needs to be positive.
00300         @exception NegativeValue: unconf_mat_ID needs to be a positive integer.
00301         @exception NegativeValue: conf_mat_ID needs to be a positive integer.
00302         @exception NegativeValue: bars_mat_ID needs to be a positive integer.
00303         @exception WrongDimension: discr_core has a length of 2.
00304         @exception WrongDimension: discr_cover has a length of 2.
00305         @exception NegativeValue: GJ needs to be positive.
00306         """
00307         # Check
00308         if ID < 1: raise NegativeValue()
00309         if b < 0: raise NegativeValue()
00310         if e < 0: raise NegativeValue()

```

```

00311         if e > b/2: raise InconsistentGeometry()
00312         if D_bars < 0: raise NegativeValue()
00313         if Ay < 0: raise NegativeValue()
00314         if n_bars < 0: raise NegativeValue()
00315         if D_hoops < 0: raise NegativeValue()
00316         if unconf_mat_ID < 1: raise NegativeValue()
00317         if conf_mat_ID < 1: raise NegativeValue()
00318         if bars_mat_ID < 1: raise NegativeValue()
00319         if len(discr_core) != 2: raise WrongDimension()
00320         if len(discr_cover) != 2: raise WrongDimension()
00321         if GJ < 0: raise NegativeValue()
00322
00323         # Arguments
00324         self.IDID = ID
00325         self.bb = b
00326         self.ee = e
00327         self.D_barsD_bars = D_bars
00328         self.AyAy = Ay
00329         self.n_barsn_bars = n_bars
00330         self.D_hoopsD_hoops = D_hoops
00331         self.unconf_mat_IDunconf_mat_ID = unconf_mat_ID
00332         self.conf_mat_IDconf_mat_ID = conf_mat_ID
00333         self.bars_mat_IDbars_mat_ID = bars_mat_ID
00334         self.discr_corediscr_core = copy(discr_core)
00335         self.discr_coverdiscr_cover = copy(discr_cover)
00336         self.alpha_ialpha_i = alpha_i
00337         self.GJGJ = GJ
00338
00339         # Initialized the parameters that are dependent from others
00340         self.section_name_tagsection_name_tag = "None"
00341         self.InitializedInitialized = False
00342         self.ReInitReInit()
00343
00344     def ReInit(self):
00345         """
00346         Implementation of the homonym abstract method.
00347         See parent class DataManagement for detailed information.
00348         """
00349         # Memebers
00350         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
00351
00352         # Parameters
00353         self.r_barsr_bars = self.bb/2 - self.ee - self.D_hoopsD_hoops - self.D_barsD_bars/2
00354         self.r_corer_core = self.bb/2 - self.ee - self.D_hoopsD_hoops/2
00355
00356         # Create the concrete core fibers
00357         core_cmd = ['patch', 'circ', self.conf_mat_IDconf_mat_ID, *self.discr_corediscr_core, 0, 0, 0,
self.r_corer_core]
00358
00359         # Create the concrete cover fibers
00360         cover_cmd = ['patch', 'circ', self.unconf_mat_IDunconf_mat_ID, *self.discr_coverdiscr_cover,
0, 0, self.r_corer_core, self.bb/2]
00361         self.fib_secfib_sec = [['section', 'Fiber', self.IDID, '-GJ', self.GJGJ],
core_cmd, cover_cmd]
00362
00363         # Create the reinforcing fibers
00364         bars_cmd = ['layer', 'circ', self.bars_mat_IDbars_mat_ID, self.n_barsn_bars, self.AyAy, 0, 0,
self.r_barsr_bars, self.alpha_ialpha_i]
00365         self.fib_secfib_sec.append(bars_cmd)
00366
00367         # Data storage for loading/saving
00368         self.UpdateStoredDataUpdateStoredData()
00369
00370
00371
00372     # Methods
00373     def UpdateStoredData(self):
00374         """
00375         Implementation of the homonym abstract method.
00376         See parent class DataManagement for detailed information.
00377         """
00378         self.data = [{"INFO_TYPE": "FibersCirc"}, # Tag for differentiating different data
[["ID", self.IDID],
["section_name_tag", self.section_name_tagsection_name_tag],
["b", self.bb],
["e", self.ee],
["r_core", self.r_corer_core],
["D_bars", self.D_barsD_bars],
["Ay", self.AyAy],
["n_bars", self.n_barsn_bars],
["r_bars", self.r_barsr_bars],
["D_hoops", self.D_hoopsD_hoops],
["alpha_i", self.alpha_ialpha_i],
["GJ", self.GJGJ],
["conf_mat_ID", self.conf_mat_IDconf_mat_ID],
["discr_core", self.discr_corediscr_core],
["unconf_mat_ID", self.unconf_mat_IDunconf_mat_ID],

```



```

00394         ["discr_cover", self.discr_coverdiscr_cover],
00395         ["bars_mat_ID", self.bars_mat_IDbars_mat_ID],
00396         ["Initialized", self.InitializedInitialized]]
00397
00398
00399     def ShowInfo(self, plot = False, block = False):
00400         """
00401         Implementation of the homonym abstract method.
00402         See parent class DataManagement for detailed information.
00403
00404         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
00405         False.
00406         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
00407         of the program everytime that a plot should pop up). Defaults to False.
00408         """
00409         print("")
00410         print("Requested info for FibersCirc, ID = {}".format(self.IDID))
00411         print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
00412         print("Base b = {} mm and concrete cover e = {} mm".format(self.bb/mm_unit, self.ee/mm_unit))
00413         print("Radius of the confined core r_core = {} mm, radius of the bars range rBars = {} mm and
00414         initial angle alpha_i = {} deg".format(self.r_corecore/mm_unit, self.rBarsrBars/mm_unit,
00415         self.alpha_ialpha_i))
00416         print("Confined material model ID = {}".format(self.conf_mat_IDconf_mat_ID))
00417         print("Unconfined material model ID = {}".format(self.unconf_mat_IDunconf_mat_ID))
00418         print("Bars material model ID = {}".format(self.bars_mat_IDbars_mat_ID))
00419         print("Discretisation in the core [number of wedges, number of rings] =
00420         {}".format(self.discr_corediscr_core))
00421         print("Discretisation in the lateral covers [number of wedges, number of rings] =
00422         {}".format(self.discr_coverdiscr_cover))
00423         print("")
00424
00425         if plot:
00426             plot_fiber_section(self.fib_secfib_sec, matcolor=['#808080', '#D3D3D3', 'k'])
00427
00428         if block:
00429             plt.show()
00430
00431     def CreateFibers(self):
00432         """
00433         Method that initialise the fiber by calling the OpenSeesPy commands.
00434         """
00435         create_fiber_section(self.fib_secfib_sec)
00436         self.InitializedInitialized = True
00437         self.UpdateStoredDataUpdateStoredData()
00438
00439
00440     class FibersCircRCCircShape(FibersCirc):
00441         """
00442         Class that is the children of FibersCirc and combine the class RCCircShape (section) to retrieve
00443         the information needed.
00444
00445         @param FibersCirc: Parent class.
00446         """
00447         def __init__(self, ID: int, section: RCCircShape, unconf_mat_ID: int, conf_mat_ID: int,
00448         bars_mat_ID: int,
00449         discr_core: list, discr_cover: list, alpha_i=0.0, GJ=0):
00450             """
00451             Constructor of the class.
00452
00453             @param ID (int): Unique fiber section ID.
00454             @param section (RCCircShape): RCCircShape section object.
00455             @param unconf_mat_ID (int): ID of material model that will be assigned to the unconfined
00456             fibers.
00457             @param conf_mat_ID (int): ID of material model that will be assigned to the confined fibers.
00458             @param bars_mat_ID (int): ID of material model that will be assigned to the reinforcing bars
00459             fibers.
00460             @param discr_core (list): List with two entries: number of subdivisions (fibers) in the
00461             circumferential direction (number of wedges),
00462             number of subdivisions (fibers) in the radial direction (number of rings) for the confined
00463             core.
00464             @param discr_cover (list): List with two entries: number of subdivisions (fibers) in the
00465             circumferential direction (number of wedges),
00466             number of subdivisions (fibers) in the radial direction (number of rings) for the
00467             unconfined cover.
00468             @param alpha_i (float, optional): Angle in deg of the first vertical rebars with respect to
00469             the y axis, counterclockwise. Defaults to 0.0.
00470             @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
00471             Defaults to 0.0, assume no torsional stiffness.
00472             """
00473             self.sectionsection = deepcopy(section)
00474             super().__init__(ID, section.b, section.e, section.D_bars, section.Ay, section.n_bars,
00475             section.D_hoops, unconf_mat_ID, conf_mat_ID, bars_mat_ID,
00476             discr_core, discr_cover, alpha_i=alpha_i, GJ=GJ)
00477             self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
00478             self.UpdateStoredDataUpdateStoredData()
00479

```

```

00464
00465 class FibersIShape(Fibers):
00466     """
00467     Class that stores functions, material properties, geometric and mechanical parameters for a steel I
    shape (non double symmetric) fiber section.
00468     Coordinates: plotting coordinte (x, y) = fiber section coordinate (z, y) = (-x, y). For more
    information, see the OpenSeesPy documentation.
00469
00470     @param Fibers: Parent abstract class.
00471     """
00472     def __init__(self, ID: int, d, bf_t, bf_b, tf_t, tf_b, tw, top_flange_mat_ID: int,
    bottom_flange_mat_ID: int, web_mat_ID: int,
00473     discr_top_flange: list, discr_bottom_flange: list, discr_web: list, GJ = 0.0):
00474         """
00475         Constructor of the class.
00476
00477         @param ID (int): Unique fiber section ID.
00478         @param d (float): Depth of the section.
00479         @param bf_t (float): Top flange's width of the section
00480         @param bf_b (float): Bottom flange's width of the section
00481         @param tf_t (float): Top flange's thickness of the section
00482         @param tf_b (float): Bottom flange's thickness of the section
00483         @param tw (float): Web's thickness of the section
00484         @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange
    fibers.
00485         @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom
    flange fibers.
00486         @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
00487         @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y)
    for the top flange.
00488         @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK
    (y) for the bottom flange.
00489         @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
    web.
00490         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
    Defaults to 0.0, assume no torsional stiffness.
00491
00492         @exception NegativeValue: ID needs to be a positive integer.
00493         @exception NegativeValue: d needs to be positive.
00494         @exception NegativeValue: bf_t needs to be positive.
00495         @exception NegativeValue: bf_b needs to be positive.
00496         @exception NegativeValue: tf_t needs to be positive.
00497         @exception NegativeValue: tf_b needs to be positive.
00498         @exception NegativeValue: tw needs to be positive.
00499         @exception NegativeValue: top_flange_mat_ID needs to be a positive integer.
00500         @exception NegativeValue: bottom_flange_mat_ID needs to be a positive integer.
00501         @exception NegativeValue: web_mat_ID needs to be a positive integer.
00502         @exception WrongDimension: discr_top_flange has a length of 2.
00503         @exception WrongDimension: discr_bottom_flange has a length of 2.
00504         @exception WrongDimension: discr_web has a length of 2.
00505         @exception NegativeValue: GJ needs to be positive.
00506         @exception InconsistentGeometry: The sum of the flanges thickness can't be bigger than d.
00507         """
00508         # Check
00509         if ID < 1: raise NegativeValue()
00510         if d < 0: raise NegativeValue()
00511         if bf_t < 0: raise NegativeValue()
00512         if bf_b < 0: raise NegativeValue()
00513         if tf_b < 0: raise NegativeValue()
00514         if tf_t < 0: raise NegativeValue()
00515         if tw < 0: raise NegativeValue()
00516         if top_flange_mat_ID < 1: raise NegativeValue()
00517         if bottom_flange_mat_ID < 1: raise NegativeValue()
00518         if web_mat_ID < 1: raise NegativeValue()
00519         if len(discr_top_flange) != 2: raise WrongDimension()
00520         if len(discr_bottom_flange) != 2: raise WrongDimension()
00521         if len(discr_web) != 2: raise WrongDimension()
00522         if GJ < 0: raise NegativeValue()
00523         if tf_t+tf_b >= d: raise InconsistentGeometry()
00524
00525         # Arguments
00526         self.IDID = ID
00527         self.dd = d
00528         self.bf_tbf_t = bf_t
00529         self.bf_bbf_b = bf_b
00530         self.tf_ttf_t = tf_t
00531         self.tf_btf_b = tf_b
00532         self.twtw = tw
00533         self.top_flange_mat_IDtop_flange_mat_ID = top_flange_mat_ID
00534         self.bottom_flange_mat_IDbottom_flange_mat_ID = bottom_flange_mat_ID
00535         self.web_mat_IDweb_mat_ID = web_mat_ID
00536         self.discr_top_flangediscr_top_flange = copy(discr_top_flange)
00537         self.discr_bottom_flangediscr_bottom_flange = copy(discr_bottom_flange)
00538         self.discr_webdiscr_web = copy(discr_web)
00539         self.GJGJ = GJ
00540
00541         # Initialized the parameters that are dependent from others

```

```

00542         self.section_name_tagsection_name_tag = "None"
00543         self.InitializedInitialized = False
00544         self.ReInitReInit()
00545
00546     def ReInit(self):
00547         """
00548         Implementation of the homonym abstract method.
00549         See parent class DataManagement for detailed information.
00550         """
00551         # Memebers
00552         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
00553
00554         # Parameters
00555         z1 = self.twtw/2
00556         y1 = (self.dd - self.tf_ttf_t - self.tf_btf_b)/2
00557
00558         # Create the flange top
00559         flange_top = [y1, -self.bf_tbf_t/2, y1+self.tf_ttf_t, self.bf_tbf_t/2]
00560         flange_top_cmd = ['patch', 'rect', self.top_flange_mat_IDtop_flange_mat_ID,
*self.discr_top_flangediscr_top_flange, *flange_top]
00561
00562         # Create the flange bottom
00563         flange_bottom = [-y1-self.tf_btf_b, -self.bf_bbf_b/2, -y1, self.bf_bbf_b/2]
00564         flange_bottom_cmd = ['patch', 'rect', self.bottom_flange_mat_IDbottom_flange_mat_ID,
*self.discr_bottom_flangediscr_bottom_flange, *flange_bottom]
00565
00566         # Create the web
00567         web = [-y1, -z1, y1, z1]
00568         web_cmd = ['patch', 'rect', self.web_mat_IDweb_mat_ID, *self.discr_webdiscr_web, *web]
00569
00570         self.fib_secfib_sec = [['section', 'Fiber', self.IDID, '-GJ', self.GJGJ],
00571                                flange_top_cmd, web_cmd, flange_bottom_cmd]
00572
00573         # Data storage for loading/saving
00574         self.UpdateStoredDataUpdateStoredData()
00575
00576
00577     # Methods
00578     def UpdateStoredData(self):
00579         """
00580         Implementation of the homonym abstract method.
00581         See parent class DataManagement for detailed information.
00582         """
00583         self.datadata = [{"INFO_TYPE", "FibersIShape"}, # Tag for differentiating different data
00584                           ["ID", self.IDID],
00585                           ["section_name_tag", self.section_name_tagsection_name_tag],
00586                           ["d", self.dd],
00587                           ["bf_t", self.bf_tbf_t],
00588                           ["bf_b", self.bf_bbf_b],
00589                           ["tf_t", self.tf_ttf_t],
00590                           ["tf_b", self.tf_btf_b],
00591                           ["tw", self.twtw],
00592                           ["GJ", self.GJGJ],
00593                           ["top_flange_mat_ID", self.top_flange_mat_IDtop_flange_mat_ID],
00594                           ["bottom_flange_mat_ID", self.bottom_flange_mat_IDbottom_flange_mat_ID],
00595                           ["web_mat_ID", self.web_mat_IDweb_mat_ID],
00596                           ["discr_top_flange", self.discr_top_flangediscr_top_flange],
00597                           ["discr_bottom_flange", self.discr_bottom_flangediscr_bottom_flange],
00598                           ["discr_web", self.discr_webdiscr_web],
00599                           ["Initialized", self.InitializedInitialized]]
00600
00601
00602     def ShowInfo(self, plot = False, block = False):
00603         """
00604         Implementation of the homonym abstract method.
00605         See parent class DataManagement for detailed information.
00606
00607         @param plot (bool, optional): Option to show the plot of the fiber. Defaults to False.
00608         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00609         """
00610         print("")
00611         print("Requested info for FibersRect, ID = {}".format(self.IDID))
00612         print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
00613         print("Depth d = {} mm and web thickness tw = {} mm".format(self.dd/mm_unit,
self.twtw/mm_unit))
00614         print("Top flange width bf_t = {} mm and thickness tf_t = {} mm".format(self.bf_tbf_t/mm_unit,
self.tf_ttf_t/mm_unit))
00615         print("Bottom flange width bf_b = {} mm and thickness tf_b = {}
mm".format(self.bf_bbf_b/mm_unit, self.tf_btf_b/mm_unit))
00616         print("Web material model ID = {}".format(self.web_mat_IDweb_mat_ID))
00617         print("Top flange material model ID = {}".format(self.top_flange_mat_IDtop_flange_mat_ID))
00618         print("Bottom flange material model ID =
{}".format(self.bottom_flange_mat_IDbottom_flange_mat_ID))
00619         print("Discretisation in the web [IJ or x/z dir, JK or y dir] =
{}".format(self.discr_webdiscr_web))

```

```

00620         print("Discretisation in the top flange [IJ or x/z dir, JK or y dir] =
{}").format(self.discr_top_flangediscr_top_flange))
00621         print("Discretisation in the bottom flange [IJ or x/z dir, JK or y dir] =
{}").format(self.discr_bottom_flangediscr_bottom_flange))
00622         print("")
00623
00624         if plot:
00625             plot_fiber_section(self.fib_secfib_sec, matcolor=['r', 'b', 'g', 'k'])
00626
00627         if block:
00628             plt.show()
00629
00630
00631     def CreateFibers(self):
00632         """
00633         Method that initialise the fiber by calling the OpenSeesPy commands.
00634         """
00635         create_fiber_section(self.fib_secfib_sec)
00636         self.InitializedInitialized = True
00637         self.UpdateStoredDataUpdateStoredData()
00638
00639
00640 class FibersIShapeSteelIShape(FibersIShape):
00641     """
00642     Class that is the children of FibersIShape and combine the class SteelIShape (section) to retrieve
the information needed.
00643
00644     @param FibersIShape: Parent class.
00645     """
00646     def __init__(self, ID: int, section: SteelIShape, top_flange_mat_ID: int, discr_top_flange: list,
discr_bottom_flange: list, discr_web: list,
00647                 GJ=0.0, bottom_flange_mat_ID = -1, web_mat_ID = -1):
00648         """
00649         Constructor of the class.
00650
00651         @param ID (int): Unique fiber section ID.
00652         @param section (SteelIShape): SteelIShape section object.
00653         @param top_flange_mat_ID (int): ID of material model that will be assigned to the top flange
fibers.
00654         @param discr_top_flange (list): List with two entries: discretisation in IJ (x/z) and JK (y)
for the top flange.
00655         @param discr_bottom_flange (list): List with two entries: discretisation in IJ (x/z) and JK
(y) for the bottom flange.
00656         @param discr_web (list): List with two entries: discretisation in IJ (x/z) and JK (y) for the
web.
00657         @param GJ (float, optional): Linear-elastic torsional stiffness assigned to the section.
Defaults to 0.0, assume no torsional stiffness.
00658         @param bottom_flange_mat_ID (int): ID of material model that will be assigned to the bottom
flange fibers.
00659             Defaults to -1, e.g. equal to top_flange_mat_ID.
00660         @param web_mat_ID (int): ID of material model that will be assigned to the web fibers.
00661             Defaults to -1, e.g. equal to top_flange_mat_ID.
00662         """
00663         self.sectionsection = deepcopy(section)
00664         if bottom_flange_mat_ID == -1: bottom_flange_mat_ID = top_flange_mat_ID
00665         if web_mat_ID == -1: web_mat_ID = top_flange_mat_ID
00666
00667         super().__init__(ID, section.d, section.bf, section.bf, section.tf, section.tf, section.tw,
top_flange_mat_ID, bottom_flange_mat_ID, web_mat_ID,
00668                         discr_top_flange, discr_bottom_flange, discr_web, GJ)
00669         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
00670         self.UpdateStoredDataUpdateStoredData()
00671
00672
00673     def plot_fiber_section(fiber_info, fill_shapes = True, matcolor=['#808080', '#D3D3D3', 'r', 'b', 'g',
'y']):
00674         """
00675         Plot fiber cross-section. Coordinate system used: plotting coordinte = (x, y), fiber section
coordinate (z, y) = (-x, y)
00676         Inspired by plot_fiber_section from ops_vis written by Seweryn Kokot.
00677
00678         @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first
list defines the fiber section: \n
['section', 'Fiber', ID, '-GJ', GJ] \n
The other lists have one of the following format (coordinate input: (y, z)!): \n
['layer', 'bar', mat_ID, A, y, z] # one bar \n
['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first
bar, J = last bar) \n
['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars
(with C = center, r = radius) \n
['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK =
b/2) \n
['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped
(starting from bottom left, counterclockwise: I, J, K, L) \n
['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri =
internal radius, re = external radius)
00687         @param fill_shapes (bool, optional): Option to fill fibers with color specified in matcolor.

```

```

Defaults to True.
00688     @param matcolor (list, optional): List of colors for various material IDs. Defaults to ['#808080',
    '#D3D3D3', 'r', 'b', 'g', 'y'].
00689
00690     Example 1: Simple rectangle with 2 rebars (D = diameter) on top (e distance from the top and from
    the lateral borders).
00691     Rectangle with first corner = I (bottom right) and second corner = K (top left); number of
    fibers = discr (list of 2)
00692     fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
00693               ['patch', 'rect', concrete_mat_ID, *discr, yI, zI, yK, zK],
00694               ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, zI-e-D/2], # left rebar
00695               ['layer', 'bar', bars_mat_ID, Ay, yI-e-D/2, -(zI-e-D/2)] # right rebar
00696
00697     Example 2: double symmetric I shape.
00698     Each rectangle (2 flanges and 1 web): first corner = I (bottom right) and second corner = K
    (top left); number of fibers = discr (list of 2)
00699     fib_sec = [['section', 'Fiber', ID, '-GJ', GJ],
00700               ['patch', 'rect', mat_ID, *discr, yI_tf, zI_tf, yK_tf, zK_tf], # top flange
00701               ['patch', 'rect', mat_ID, *discr, yI_bf, zI_bf, yK_bf, zK_bf], # bottom flange
00702               ['patch', 'rect', mat_ID, *discr, yI_w, zI_w, yK_w, zK_w]] # web
00703     """
00704
00705     mat_to_col = {}
00706     fig, ax = plt.subplots()
00707     ax.grid(False)
00708
00709     for item in fiber_info:
00710         if item[0] == 'section':
00711             fib_ID = item[2]
00712         if item[0] == 'layer':
00713             matID = item[2]
00714             mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
00715             if item[1] == 'bar':
00716                 As = item[3]
00717                 Iy = item[4]
00718                 Iz = item[5]
00719                 r = np.sqrt(As / np.pi)
00720                 bar = Circle((-Iz, Iy), r, ec='k', fc='k', zorder=10)
00721                 ax.add_patch(bar)
00722             if item[1] == 'straight':
00723                 n_bars = item[3]
00724                 As = item[4]
00725                 Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
00726                 r = np.sqrt(As / np.pi)
00727                 Y = np.linspace(Iy, Jy, n_bars)
00728                 Z = np.linspace(Iz, Jz, n_bars)
00729                 for zi, yi in zip(Z, Y):
00730                     bar = Circle((-zi, yi), r, ec='k', fc=mat_to_col[matID], zorder=10)
00731                     ax.add_patch(bar)
00732             if item[1] == 'circ':
00733                 n_bars, As = item[3], item[4]
00734                 yC, zC, r = item[5], item[6], item[7]
00735                 if len(item) > 8:
00736                     a0_deg = item[8]
00737                 else:
00738                     a0_deg = 0.
00739                 a1_deg = 360. - 360./n_bars + a0_deg
00740                 if len(item) > 9: a1_deg = item[9]
00741
00742                 a0_rad, a1_rad = np.pi * a0_deg / 180., np.pi * a1_deg / 180.
00743                 r_bar = np.sqrt(As / np.pi)
00744                 thetas = np.linspace(a0_rad, a1_rad, n_bars)
00745                 Y = yC + r * np.cos(thetas)
00746                 Z = zC + r * np.sin(thetas)
00747                 for zi, yi in zip(Z, Y):
00748                     bar = Circle((-zi, yi), r_bar, ec='k', fc=mat_to_col[matID], zorder=10)
00749                     ax.add_patch(bar)
00750
00751             if (item[0] == 'patch' and (item[1] == 'quad' or item[1] == 'quadr' or
00752                                         item[1] == 'rect')):
00753                 matID, nIJ, nJK = item[2], item[3], item[4]
00754                 mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
00755
00756             if item[1] == 'quad' or item[1] == 'quadr':
00757                 Iy, Iz, Jy, Jz = item[5], item[6], item[7], item[8]
00758                 Ky, Kz, Ly, Lz = item[9], item[10], item[11], item[12]
00759
00760             if item[1] == 'rect':
00761                 Iy, Iz, Ky, Kz = item[5], item[6], item[7], item[8]
00762                 Jy, Jz, Ly, Lz = Iy, Kz, Ky, Iz
00763                 # check order of definition
00764                 if Kz-Iz < 0 or Ky-Iy < 0: print("!!!!!! WARNING !!!!!!! The fiber is not defined
00765 bottom right, top left")
00766
00767                 # check for convexity (vector products)
00768                 outIJxIK = (Jy-Iy)*(Kz-Iz) - (Ky-Iy)*(Jz-Iz)

```

```

00769         outIKxIL = (Ky-Iy)*(Lz-Iz) - (Ly-Iy)*(Kz-Iz)
00770         # check if I, J, L points are colinear
00771         outIJxIL = (Jy-Iy)*(Lz-Iz) - (Ly-Iy)*(Jz-Iz)
00772         # outJKxJL = (Ky-Jy)*(Lz-Jz) - (Ly-Jy)*(Kz-Jz)
00773
00774         if -outIJxIK <= 0 or -outIKxIL <= 0 or -outIJxIL <= 0:
00775             print('!!!!!!! WARNING !!!!!!! Patch quad is non-convex or non-counter-clockwise
defined or has at least 3 colinear points in line')
00776
00777         IJz, IJy = np.linspace(Iz, Jz, nIJ+1), np.linspace(Iy, Jy, nIJ+1)
00778         JKz, JKy = np.linspace(Jz, Kz, nJK+1), np.linspace(Jy, Ky, nJK+1)
00779         LKz, LKy = np.linspace(Lz, Kz, nIJ+1), np.linspace(Ly, Ky, nIJ+1)
00780         ILz, ILy = np.linspace(Iz, Lz, nJK+1), np.linspace(Iy, Ly, nJK+1)
00781
00782         if fill_shapes:
00783             Z = np.zeros((nIJ+1, nJK+1))
00784             Y = np.zeros((nIJ+1, nJK+1))
00785
00786             for j in range(nIJ+1):
00787                 Z[j, :] = np.linspace(IJz[j], LKz[j], nJK+1)
00788                 Y[j, :] = np.linspace(IJy[j], LKy[j], nJK+1)
00789
00790             for j in range(nIJ):
00791                 for k in range(nJK):
00792                     zy = np.array([[ -Z[j, k], Y[j, k]],
00793                                   [ -Z[j, k+1], Y[j, k+1]],
00794                                   [ -Z[j+1, k+1], Y[j+1, k+1]],
00795                                   [ -Z[j+1, k], Y[j+1, k]]])
00796                     poly = Polygon(zy, True, ec='k', fc=mat_to_col[matID])
00797                     ax.add_patch(poly)
00798
00799             else:
00800                 # horizontal lines
00801                 for az, bz, ay, by in zip(IJz, LKz, IJy, LKy):
00802                     plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)
00803
00804                 # vertical lines
00805                 for az, bz, ay, by in zip(JKz, ILz, JKy, ILy):
00806                     plt.plot([-az, -bz], [ay, by], 'b-', zorder=1)
00807
00808         if item[0] == 'patch' and item[1] == 'circ':
00809             matID, nc, nr = item[2], item[3], item[4]
00810             mat_to_col = __assignColorToMat(matID, mat_to_col, matcolor)
00811
00812             yC, zC, ri, re = item[5], item[6], item[7], item[8]
00813             if len(item) > 9:
00814                 a0 = item[9]
00815             else:
00816                 a0 = 0.
00817             a1 = 360. + a0
00818             if len(item) > 10: a1 = item[10]
00819
00820             dr = (re - ri) / nr
00821             dth = (a1 - a0) / nc
00822
00823             for j in range(nr):
00824                 rj = ri + j * dr
00825                 rj1 = rj + dr
00826
00827                 for i in range(nc):
00828                     thi = a0 + i * dth
00829                     th1l = thi + dth
00830                     wedge = Wedge((-zC, yC), rj1, thi, th1l, width=dr, ec='k', #Seweryn Kokot: (yC,
-zC), wrong??
00831                                     lw=1, fc=mat_to_col[matID])
00832                     ax.add_patch(wedge)
00833             ax.set(xlabel='x dimension {}'.format(length_unit), ylabel='y dimension
[{}].format(length_unit),
00834                   title='Fiber section (ID = {})'.format(fib_ID))
00835             ax.axis('equal')
00836
00837     def __assignColorToMat(matID: int, mat_to_col: dict, matcolor: list):
00838         """
00839         PRIVATE FUNCTION. Used to assign different colors for each material model assign to the fiber
section.
00840
00841         @param matID (int): ID of the material model.
00842         @param mat_to_col (dict): Dictionary to check with material model has which color.
00843         @param matcolor (list): List of colors.
00844
00845         @returns dict: Updated dictionary.
00846         """
00847         if not matID in mat_to_col:
00848             if len(mat_to_col) >= len(matcolor):
00849                 print("Warning: not enough colors defined for fiber section plot (white used)")
00850                 mat_to_col[matID] = 'w'

```

```

00852         else:
00853             mat_to_col[matID] = matcolor[len(mat_to_col)]
00854         return mat_to_col
00855
00856
00857 def create_fiber_section(fiber_info):
00858     """
00859     Initialise fiber cross-section with OpenSeesPy commands.
00860     For examples, see plot_fiber_section.
00861     Inspired by fib_sec_list_to_cmds from ops_vis written by Seweryn Kokot
00862
00863     @param fiber_info (list): List of lists (be careful with the local coordinate system!). The first
00864     list defines the fiber section: \n
00865     ['section', 'Fiber', ID, '-GJ', GJ] \n
00866     The other lists have one of the following format (coordinate input: (y, z)!): \n
00867     ['layer', 'bar', mat_ID, A, y, z] # one bar \n
00868     ['layer', 'straight', mat_ID, n_bars, A, yI, zI, yJ, zJ] # line range of bars (with I = first
00869     bar, J = last bar) \n
00870     ['layer', 'circ', mat_ID, n_bars, A, yC, zC, r, (a0_deg), (a1_deg)] # circular range of bars
00871     (with C = center, r = radius) \n
00872     ['patch', 'rect', mat_ID, *discr, -yI, zI, yK, -zK] # rectangle (with yI = yK = d/2; zI = zK =
00873     b/2) \n
00874     ['patch', 'quad', mat_ID, *discr, yI, zI, yJ, zJ, yK, zK, yL, zL] # quadrilateral shaped
00875     (starting from bottom left, counterclockwise: I, J, K, L) \n
00876     ['patch', 'circ', mat_ID, *discr, yC, zC, ri, re, (a0), (a1)] # (with C = center, ri =
00877     internal radius, re = external radius)
00878     """
00879     for dat in fiber_info:
00880         if dat[0] == 'section':
00881             fib_ID, GJ = dat[2], dat[4]
00882             section('Fiber', fib_ID, 'GJ', GJ)
00883
00884         if dat[0] == 'layer':
00885             mat_ID = dat[2]
00886             if dat[1] == 'straight':
00887                 n_bars = dat[3]
00888                 As = dat[4]
00889                 Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
00890                 layer('straight', mat_ID, n_bars, As, Iy, Iz, Jy, Jz)
00891             if dat[1] == 'bar':
00892                 As = dat[3]
00893                 Iy = dat[4]
00894                 Iz = dat[5]
00895                 fiber(Iy, Iz, As, mat_ID)
00896                 # layer('straight', mat_ID, 1, As, Iy, Iz, Iy, Iz)
00897             if dat[1] == 'circ':
00898                 n_bars, As = dat[3], dat[4]
00899                 yC, zC, r = dat[5], dat[6], dat[7]
00900                 if len(dat) > 8:
00901                     a0_deg = dat[8]
00902                 else:
00903                     a0_deg = 0.
00904                     a1_deg = 360. - 360./n_bars + a0_deg
00905                 if len(dat) > 9: a1_deg = dat[9]
00906                 layer('circ', mat_ID, n_bars, As, yC, zC, r, a0_deg, a1_deg)
00907
00908         if dat[0] == 'patch':
00909             mat_ID = dat[2]
00910             nIJ = dat[4]
00911             nJK = dat[3]
00912
00913             if dat[1] == 'quad' or dat[1] == 'quadr':
00914                 Iy, Iz, Jy, Jz = dat[5], dat[6], dat[7], dat[8]
00915                 Ky, Kz, Ly, Lz = dat[9], dat[10], dat[11], dat[12]
00916                 patch('quad', mat_ID, nIJ, nJK, Iy, Iz, Jy, Jz, Ky, Kz,
00917                     Ly, Lz)
00918
00919             if dat[1] == 'rect':
00920                 Iy, Iz, Ky, Kz = dat[5], dat[6], dat[7], dat[8]
00921                 patch('rect', mat_ID, nIJ, nJK, Iy, Iz, Ky, Kz)
00922                 # patch('rect', mat_ID, nIJ, nJK, Iy, Kz, Ky, Iz)
00923
00924             if dat[1] == 'circ':
00925                 mat_ID, nc, nr = dat[2], dat[3], dat[4]
00926                 yC, zC, ri, re = dat[5], dat[6], dat[7], dat[8]
00927                 if len(dat) > 9:
00928                     a0 = dat[9]
00929                 else:
00930                     a0 = 0.
00931                     a1 = 360. + a0
00932                 if len(dat) > 10: a1 = dat[10]
00933                 patch('circ', mat_ID, nc, nr, yC, zC, ri, re, a0, a1)
00934

```

## 8.13 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/↵ FunctionalFeatures.py File Reference

### Classes

- class [IDGenerator](#)  
*Class that manage the ID generation.*

### Namespaces

- namespace [FunctionalFeatures](#)  
*Module with useful functions (discretise curves, ID conventions, etc)  
Carmine Schipani, 2021.*

### Functions

- def [DiscretizeLinearly](#) (np.ndarray LP, int discr, plot=False, block=False, show\_original\_LP=True)  
*This function discretize the curve 'LP' given adding the number of point given by 'discr' between every point (linearly).*
- def [DiscretizeLoadProtocol](#) (np.ndarray SDR\_LP, np.ndarray nr\_cycles\_LP, int discr\_first\_cycle, plot=False, block=False, show\_original\_peaks=True)  
*Discretized a cyclic load protocol keeping a similar discretisation step throughout the different cycles and keeping in the output the extremes (peaks).*
- def [GridIDConvention](#) (int pier\_axis, int floor\_axis, max\_pier=-1, max\_floor=-1)  
*Function used to construct the ID of the nodes in the grid (first nodes that define the geometry of the model).*
- def [IDConvention](#) (int prefix, int suffix, n\_zeros\_between=0)  
*Function used to construct IDs for elements and offgrid nodes.*
- def [NodesOrientation](#) (int iNode\_ID, int jNode\_ID)  
*Function that finds the orientation of the vector with direction 'jNode\_ID'-'iNode\_ID'.*
- def [OffsetNodeIDConvention](#) (int node\_ID, str orientation, str position\_i\_or\_j)  
*Function used to add node on top of existing ones in the extremes of members with springs.*
- def [plot\\_member](#) (list element\_array, member\_name="Member name not defined", show\_element\_ID=True, show\_node\_ID=True)  
*Function that plots a set of elements.*
- def [plot\\_nodes](#) (list nodes\_array, name="Not defined", show\_node\_ID=True)  
*Function that plots a set of nodes.*
- def [ProgressingPercentage](#) (max\_iter, int i, int next\_step, step=10)  
*Function that shows the progressing percentage of an iterative process.*

## 8.14 FunctionalFeatures.py

[Go to the documentation of this file.](#)

```
00001 """Module with useful functions (discretise curves, ID conventions, etc) \n
00002 Carmine Schipani, 2021
00003 """
00004
00005 import math
00006 import numpy as np
00007 import matplotlib.pyplot as plt
00008 import openseespy.postprocessing.internal_plotting_functions as ipltf
00009 from openseespy.opensees import *
00010 from OpenSeesPyAssistant.ErrorHandling import *
00011 from OpenSeesPyAssistant.Units import *
```



```

00012
00013
00014 def ProgressingPercentage(max_iter, i: int, next_step: int, step = 10):
00015     """
00016     Function that shows the progressing percentage of an iterative process.
00017
00018     @param max_iter (int): Maximal number of iterations
00019     @param i (int): Current iteration
00020     @param next_step (int): Next step of the percentage (set to 0 for the first iteration and then use
the return parameter)
00021     @param step (int, optional): Size of the step (should be a fraction of 100). Defaults to 10.
00022
00023     @returns int: The updated next step
00024     """
00025     if i*100.0/(max_iter-1) >= next_step:
00026         print("The progression is {}".format(next_step))
00027         return next_step + step
00028
00029     return next_step
00030
00031
00032 def DiscretizeLoadProtocol(SDR_LP: np.ndarray, nr_cycles_LP: np.ndarray, discr_first_cycle: int, plot
= False, block = False, show_original_peaks = True):
00033     """
00034     Discretized a cyclic load protocol keeping a similar discretisation step throughout the different
cycles and keeping in the output the extremes (peaks).
00035
00036     @param SDR_LP (np.ndarray): Array (1 dimension) that stores the peaks of the cycles.
00037     They needs to be only the positive peaks, beacuse this function will use them as the extreme
for each cycle.
00038     @param nr_cycles_LP (np.ndarray): Array (1 dimension) that stores the number of cycles for every
extreme declared in 'SDR_LP' and its countepart negative.
00039     They need to be positive integers.
00040     @param discr_first_cycle (int): The number of points from peak to peak (counting the two peaks) in
the first cycle. It should be odd.
00041     @param plot (bool, optional): Option to show the plot of the discretized (and also the original
peaks). Defaults to False.
00042     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of
the program everytime that a plot should pop up). Defaults to False.
00043     @param show_original_peaks (bool, optional): Option to show the original peaks to check if the
discretized curve is correct.
00044     The argument plot need to be True. Defaults to True.
00045
00046     @exception WrongDimension: SDR_LP and nr_cycles_LP need to be of same length.
00047     @exception NegativeValue: SDR_LP needs to have only positive integers.
00048     @exception NegativeValue: nr_cycles_LP needs to have only positive integers.
00049     @exception NegativeValue: discr_first_cycle needs to be a positive integer.
00050
00051     @returns np.array: Array (1 dimension) that stores the new discretized load protocol curve.
00052     """
00053     if np.size(SDR_LP) != np.size(nr_cycles_LP): raise WrongDimension()
00054     if any(col < 0 for col in SDR_LP): raise NegativeValue()
00055     if any(col < 0 for col in nr_cycles_LP): raise NegativeValue()
00056     if discr_first_cycle < 0: raise NegativeValue()
00057
00058     if discr_first_cycle % 2 == 0:
00059         discr_first_cycle = discr_first_cycle + 1
00060     discr_factor = discr_first_cycle / (SDR_LP[0]*2)
00061     discretized_LP = [0.0]
00062     x_val = []
00063     skip_x = 0
00064     for i in range(np.size(SDR_LP)):
00065         discr_i = math.ceil(discr_factor*SDR_LP[i]*2)-1;
00066         if discr_i % 2 == 0:
00067             discr_i = discr_i + 1
00068         length_tmp = int((discr_i+1)/2)
00069         tmp_up = np.linspace(0.0, SDR_LP[i], length_tmp)
00070         tmp_down = np.linspace(SDR_LP[i], 0.0, length_tmp)
00071         for j in range(int(nr_cycles_LP[i])):
00072             discretized_LP = np.append(discretized_LP, tmp_up[1:length_tmp])
00073             discretized_LP = np.append(discretized_LP, tmp_down[1:length_tmp])
00074             discretized_LP = np.append(discretized_LP, -tmp_up[1:length_tmp])
00075             discretized_LP = np.append(discretized_LP, -tmp_down[1:length_tmp])
00076         # for check of original peaks
00077         x_val.append(length_tmp-1+skip_x)
00078         skip_x = (length_tmp-1)*(4*(nr_cycles_LP[i]-1)+3)+x_val[-1]
00079
00080
00081     if plot:
00082         fig, ax = plt.subplots()
00083         ax.plot(discretized_LP, '-r', label='Discretised LP')
00084
00085         ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
00086               title='Discretized loading protocol')
00087         ax.grid()
00088
00089     if show_original_peaks:

```

```

00090         ax.plot(x_val, SDR_LP, 'ob', label='Original LP')
00091         ax.legend()
00092
00093         if block:
00094             plt.show()
00095
00096         return discretized_LP
00097
00098
00099 def DiscretizeLinearly(LP: np.ndarray, discr: int, plot = False, block = False, show_original_LP =
True):
00100     """
00101     This function discretize the curve 'LP' given adding the number of point given by 'discr' between
every point (linearly).
00102
00103     @param LP (np.ndarray): Array (1 dimension) that stores the curve that needs to be discretized
00104     @param discr (int): The number of points to add between every two points of 'LP' (linearly)
00105     @param plot (bool, optional): Option to show the plot of the discretized (and also the original
LP). Defaults to False.
00106     @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop of
the program everytime that a plot should pop up). Defaults to False.
00107     @param show_original_LP (bool, optional): Option to show the original LP to check if the
discretized curve is correct. Defaults to True.
00108
00109     @returns np.ndarray: Array (1 dimension) that stores the new discretized load protocol.
00110     """
00111
00112     #TODO: check discr nonnegative int and LP 1 dimension
00113
00114     # Define the new discretized LP
00115     length = 1 + (np.size(LP)-1) * (discr+1)
00116     discr_LP = np.zeros(length)
00117
00118     # Performa manually the first iteration
00119     yprev = LP[0]
00120     x = [0, 1]
00121     discr_LP[0] = yprev
00122     iter = 0
00123
00124     # add the other points and the discretized ones
00125     for ynext in LP[1:]:
00126         y = [yprev, ynext]
00127
00128         # Compute new points
00129         xnew = np.linspace(x[0], x[1], discr+2)
00130         ynew = np.interp(xnew[1:], x, y)
00131
00132         # Add to the recording vector discr_LP
00133         index = np.array(np.arange(discr+1)+1+iter)
00134         discr_LP[index] = ynew
00135
00136         # Prepare for next iteration
00137         yprev = ynext
00138         iter = iter + discr + 1
00139
00140     if plot:
00141         fig, ax = plt.subplots()
00142         ax.plot(discr_LP, '-r', label='Discretised LP')
00143
00144         ax.set(xlabel='Step number [-]', ylabel='Unit of the loading protocol',
00145               title='Discretized loading protocol')
00146         ax.grid()
00147
00148         if show_original_LP:
00149             x_val = np.arange(0, np.size(discr_LP), discr+1)
00150             ax.plot(x_val, LP, 'ob', label='Original LP')
00151             ax.legend()
00152
00153         if block:
00154             plt.show()
00155
00156     return discr_LP
00157
00158
00159 def GridIDConvention(pier_axis: int, floor_axis: int, max_pier = -1, max_floor = -1):
00160     """
00161     Function used to construct the ID of the nodes in the grid (first nodes that define the geometry
of the model).
00162     The conventional grid node ID is xy, with x = the pier position 'pier_axis'; y = the floor
position 'floor_axis'.
00163
00164     @param pier_axis (int): The pier (or x) postion of the node.
00165     @param floor_axis (int): The floor (or y) position of the node.
00166     @param max_pier (int, optional): Maximal pier position of the model (used to identify the number
of digits).
00167     Defaults to -1, e.g. taken equal of pier_axis.
00168     @param max_floor (int, optional): Maximal floor position of the model (used to identify the number

```

```

of digits).
00169     Defaults to -1, e.g. taken equal of floor_axis.
00170
00171     @exception NameError: Work In Progress: only 9 floors/bays.
00172     @exception NegativeValue: The argument pier_axis needs to be a positive integer.
00173     @exception NegativeValue: The argument floor_axis needs to be a positive integer.
00174     @exception NegativeValue: The argument max_pier needs to be a positive integer if different from
-1.
00175     @exception NegativeValue: The argument max_floor needs to be a positive integer if different from
-1.
00176     @exception WrongArgument: The argument max_pier need to be equal or bigger to pier_axis
00177     @exception WrongArgument: The argument max_floor need to be equal or bigger to floor_axis
00178
00179     @returns int: The grid node ID
00180     """
00181     # Convention:
00182     #     GridNodeID:      xy          with x = pier, y = floor
00183     if pier_axis > 9 or floor_axis > 9 or max_pier > 9 or max_floor > 9: raise NameError("WIP: maximal
9 floors or bays")
00184     max_pier = pier_axis if max_pier == -1 else max_pier
00185     max_floor = floor_axis if max_floor == -1 else max_floor
00186
00187     if pier_axis < 0: raise NegativeValue()
00188     if floor_axis < 0: raise NegativeValue()
00189     if max_pier < 0: raise NegativeValue()
00190     if max_floor < 0: raise NegativeValue()
00191     if max_pier < pier_axis: raise WrongArgument()
00192     if max_floor < floor_axis: raise WrongArgument()
00193
00194     max_x_digits = int(math.log10(max_pier))+1
00195     max_y_digits = int(math.log10(max_floor))+1
00196
00197     # return 10**(max_x_digits+max_y_digits) + pier_axis*10**max_y_digits + floor_axis # with 1 as
prefix (to consider more than one digit per axis, but exceed max ID)
00198     return pier_axis*10**max_y_digits + floor_axis
00199
00200
00201 def IDConvention(prefix: int, suffix: int, n_zeros_between = 0):
00202     """
00203     Function used to construct IDs for elements and offgrid nodes.
00204     It appends to a positive integer number 'prefix' a number of zeros 'n_zeros_between' and at last
another positive integer 'suffix'.
00205     The conventional element ID is xy(a)x'y'(a') with xya = the node ID in pier x, floor y and offgrid
parameter a (optional);
00206     x'y'a' = the node ID in pier x', floor y' and offgrid parameter a' (optional).
00207     For more information on x and y, see GridIDConvention; for more information on a, see
OffsetNodeIDConvention.
00208
00209     @param prefix (int): Prefix of the new ID. For a vertical element it should be the left node ID;
00210     for an horizontal one it should be the bottom node.
00211     @param suffix (int): Suffix of the new ID. For a vertical element it should be the right node ID;
00212     for an horizontal one it should be the top node.
00213     @param n_zeros_between (int, optional): Number of zeros to add between the two nodes. Defaults to
0.
00214
00215     @exception NegativeValue: The argument prefix needs to be a positive integer.
00216     @exception NegativeValue: The argument suffix needs to be a positive integer.
00217     @exception NegativeValue: The argument n_zeros_between needs to be a positive integer.
00218
00219     @returns int: The combined ID
00220     """
00221     # Convention:
00222     #     ElementID:      xy(a)x'y'(a')    with xy(a) = NodeID i and x'y'(a') = NodeID j
00223     #     TrussID:        xy(a)x'y'(a')    with xy(a) = NodeID i and x'y'(a') = NodeID j
00224     #     Spring:         xy(a)x'y'(a')    with xy(a) = NodeID i and x'y'(a') = NodeID j
00225     if prefix < 0: raise NegativeValue()
00226     if suffix < 0: raise NegativeValue()
00227     if n_zeros_between < 0: raise NegativeValue()
00228
00229     return int(str(prefix*10**n_zeros_between) + str(suffix))
00230
00231
00232 def OffsetNodeIDConvention(node_ID: int, orientation: str, position_i_or_j: str):
00233     """
00234     Function used to add node on top of existing ones in the extremes of members with springs.
00235
00236     @param node_ID (int): Node that we refer to.
00237     @param orientation (str): Orientation of the member. Can be 'vertical' or 'horizontal'.
00238     @param position_i_or_j (str): Position at the start 'i' (left or bottom)
00239     or at the end 'j' (right or top) of 'node_ID' in the member.
00240
00241     @exception NegativeValue: The argument node_ID needs to be a positive integer.
00242     @exception WrongArgument: The argument position_i_or_j needs to be 'i' or 'j'
00243     @exception WrongArgument: The argument orientation needs to be 'vertical' or 'horizontal'
00244
00245     @returns int: The combined ID
00246     """

```

```

00247     # Convention:
00248     #     o xy          GridNodeID:      xy          with x = pier, y = floor
00249     #     o xy7        AdditionalNodeID:  xya          with x = pier, y = floor, a:  o xy  xy2 o-----o x'y3
00250 x'y o |
00251     #     PanelZoneNodeID:  xy(a)a      see MemberModel for the panel zone ID convention
00252     #
00253     #     o xy'6
00254     #     o xy'
00254     if node_ID < 1: raise NegativeValue()
00255     if position_i_or_j != "i" and position_i_or_j != "j": raise WrongArgument()
00256
00257     if orientation == "vertical":
00258         if position_i_or_j == "i":
00259             return IDConvention(node_ID, 6)
00260         else:
00261             return IDConvention(node_ID, 7)
00262     elif orientation == "horizontal":
00263         if position_i_or_j == "i":
00264             return IDConvention(node_ID, 2)
00265         else:
00266             return IDConvention(node_ID, 3)
00267     else: raise WrongArgument()
00268
00269
00270 def NodesOrientation(iNode_ID: int, jNode_ID: int):
00271     """
00272     Function that finds the orientation of the vector with direction 'jNode_ID"iNode_ID'.
00273     If the the nodes are on top of each other, the function returns 'zero_length'.
00274
00275     @param iNode_ID (int): Node i.
00276     @param jNode_ID (int): Node j.
00277
00278     @exception NegativeValue: The argument iNode_ID needs to be a positive integer.
00279     @exception NegativeValue: The argument jNode_ID needs to be a positive integer.
00280
00281     @returns str: The orientation of the vector.
00282     """
00283     if iNode_ID < 1: raise NegativeValue()
00284     if jNode_ID < 1: raise NegativeValue()
00285
00286     iNode = np.array(nodeCoord(iNode_ID))
00287     jNode = np.array(nodeCoord(jNode_ID))
00288     if abs(iNode[0]-jNode[0]) + abs(iNode[1]-jNode[1]) == 0:
00289         return "zero_length"
00290     elif abs(iNode[0]-jNode[0]) < abs(iNode[1]-jNode[1]):
00291         return "vertical"
00292     else:
00293         return "horizontal"
00294
00295
00296 def plot_member(element_array: list, member_name = "Member name not defined", show_element_ID = True,
00297 show_node_ID = True):
00298     """
00299     Function that plots a set of elements. It can be used to check the correctness of a part of the
00300     model or of a member.
00301     If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from
00302     openseespy.postprocessing.Get_Rendering. \n
00303     Inspired by plot_model written by Anurag Upadhyay and Christian Slotboom.
00304
00305     @param element_array (list): An array (list of lists of one dimensions and length = 3) that store
00306     the element and nodes IDs.
00307     An element is stored in one list with 3 entries: the element ID, node i ID and node j ID.
00308     @param member_name (str, optional): The name of what is plotted. Defaults to "Member name not
00309     defined".
00310     @param show_element_ID (bool, optional): Option to show the element IDs. Defaults to True.
00311     @param show_node_ID (bool, optional): Option to show the nodes IDs. Defaults to True.
00312
00313     @exception WrongDimension: element_array needs to be non-empty.
00314     @exception WrongDimension: The number of entries in the lists inside the argument element_array
00315     need to be 3.
00316
00317     @returns matplotlib.axes._subplots.AxesSubplo: The figure's wrapprr, useful to customise the plot
00318     (change axes label, etc).
00319     """
00320     if len(element_array) == 0: raise WrongArgument()
00321     if len(element_array[0]) != 3: raise WrongDimension()
00322
00323     node_style = {'color':'black', 'marker':'o', 'facecolor':'black', 'linewidth':0.}
00324     node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
00325     track_node = {}

```

```

00320     if show_element_ID:
00321         show_e_ID = 'yes'
00322     else:
00323         show_e_ID = 'no'
00324
00325     fig = plt.figure()
00326     ax = fig.add_subplot(1,1,1)
00327
00328     for ele in element_array:
00329         eleTag = int(ele[0])
00330         Nodes =ele[1:]
00331
00332         if len(Nodes) == 2:
00333             # 2D element
00334             iNode = np.array(nodeCoord(Nodes[0]))
00335             jNode = np.array(nodeCoord(Nodes[1]))
00336             ipltf._plotBeam2D(iNode, jNode, ax, show_e_ID, eleTag, "solid")
00337             ax.scatter(*iNode, **node_style)
00338             ax.scatter(*jNode, **node_style)
00339             if show_node_ID:
00340                 if abs(sum(iNode - jNode)) > 1e-6:
00341                     # beam-col
00342                     __plt_node(Nodes[0], track_node, iNode, ax, node_text_style)
00343                     __plt_node(Nodes[1], track_node, jNode, ax, node_text_style, h_align='right',
v_align='bottom')
00344             else:
00345                 # zerolength
00346                 __plt_node(Nodes[0], track_node, iNode, ax, node_text_style, h_align='right')
00347                 __plt_node(Nodes[1], track_node, jNode, ax, node_text_style)
00348         else:
00349             print("Too many nodes in this elemnet (see shell elements)")
00350
00351     ax.set_xlabel('x [{}].format(length_unit))
00352     ax.set_ylabel('y [{}].format(length_unit))
00353     plt.title("Visualisation of: {}".format(member_name))
00354     plt.axis('equal')
00355     return ax
00356
00357
00358 def plot_nodes(nodes_array: list, name = "Not defined", show_node_ID = True):
00359     """
00360     Function that plots a set of nodes. It can be used to check the correctness of the model's
00361     geometry.
00362     If the entire model need to be plotted, use instead 'plot_model("nodes", "elements")' from
00363     openseespy.postprocessing.Get_Rendering.
00364
00365     @param nodes_array (list): List of 1 dimension with the IDs of the nodes to be displayed.
00366     @param name (str, optional): Name that describe what the plot will show. Defaults to "Not
00367     defined".
00368     @param show_node_ID (bool, optional): Option to show the node IDs. Defaults to True.
00369
00370     @exception WrongArgument: nodes_array needs to be non-empty.
00371
00372     @returns (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper, useful to customise the
00373     plot (change axes label, etc).
00374     """
00375     if len(nodes_array) == 0: raise WrongArgument()
00376
00377     node_style = {'color':'black', 'marker':'o', 'facecolor':'black', 'linewidth':0.}
00378     node_text_style = {'fontsize':8, 'fontweight':'regular', 'color':'green'}
00379     track_node = {}
00380
00381     fig = plt.figure()
00382     ax = fig.add_subplot(1,1,1)
00383
00384     for node_ID in nodes_array:
00385         node_xy = np.array(nodeCoord(node_ID))
00386         ax.scatter(*node_xy, **node_style)
00387         if show_node_ID:
00388             __plt_node(node_ID, track_node, node_xy, ax, node_text_style)
00389
00390     ax.set_xlabel('x [{}].format(length_unit))
00391     ax.set_ylabel('y [{}].format(length_unit))
00392     plt.title("Visualisation of: {}".format(name))
00393     plt.axis('equal')
00394     return ax
00395
00396 def __plt_node(nodeID: int, track_node: dict, NodeXY, ax, node_text_style, x_off = 0, y_off = 0,
h_align = 'left', v_align='top'):
00397     """
00398     PRIVATE FUNCTION. Used to plot the nodes in a controlled way (no repetition, position of the IDs,
00399     text style).
00400
00401     @param nodeID (int): The ID of the node.
00402     @param track_node (dict): A dictionary used to avoid plotting a node multiple times.
00403     @param NodeXY (list): List of dimension 1, length 2 with the position of the node.

```

```

00400     @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
00401     @param node_text_style (dict): Dictionary for the text style.
00402     @param x_off (int, optional): Offset in x direction. Defaults to 0.
00403     @param y_off (int, optional): Offset in y direction. Defaults to 0.
00404     @param h_align (str, optional): Horizontal alignment ('left' or 'right'). Defaults to 'left'.
00405     @param v_align (str, optional): Vertical alignment ('center', 'top' or 'bottom'). Defaults to
'top'.
00406     """
00407     if not nodeID in track_node:
00408         track_node[nodeID] = True
00409         ax.text(NodeXY[0]+x_off, NodeXY[1]+y_off, nodeID,**node_text_style,
horizontalalignment=h_align, verticalalignment=v_align)
00410
00411
00412 class IDGenerator():
00413     """Class that manage the ID generation.
00414     USE ONLY IF EVERY NODE IS DEFINED BY THE USER (because the OpenSeesPyAssistant modules use the
convention defined in the functions above).
00415     """
00416     def __init__(self):
00417         """The class constructor.
00418         """
00419         self.current_node_IDcurrent_node_ID = 0
00420         self.current_element_IDcurrent_element_ID = 0
00421         self.current_mat_IDcurrent_mat_ID = 0
00422         self.current_fiber_IDcurrent_fiber_ID = 0
00423
00424     def GenerateIDNode(self):
00425         """
00426         Method that generate a unique node ID.
00427
00428         @returns int: The node ID.
00429         """
00430         self.current_node_IDcurrent_node_ID = self.current_node_IDcurrent_node_ID + 1
00431         return self.current_node_IDcurrent_node_ID
00432
00433     def GenerateIDElement(self):
00434         """
00435         Method that generate a unique element ID.
00436
00437         @returns int: The element ID.
00438         """
00439         self.current_element_IDcurrent_element_ID = self.current_element_IDcurrent_element_ID + 1
00440         return self.current_element_IDcurrent_element_ID
00441
00442     def GenerateIDMat(self):
00443         """
00444         Method that generate a unique material ID.
00445
00446         @returns int: The material ID.
00447         """
00448         self.current_mat_IDcurrent_mat_ID = self.current_mat_IDcurrent_mat_ID + 1
00449         return self.current_mat_IDcurrent_mat_ID
00450
00451     def GenerateIDFiber(self):
00452         """
00453         Method that generate a unique fiber ID.
00454
00455         @returns int: The fiber ID.
00456         """
00457         self.current_fiber_IDcurrent_fiber_ID = self.current_fiber_IDcurrent_fiber_ID + 1
00458         return self.current_fiber_IDcurrent_fiber_ID
00459
00460

```

## 8.15 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/↵ GeometryTemplate.py File Reference

### Namespaces

- namespace [GeometryTemplate](#)

*Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).*

## Functions

- def `DefineFrameNodes` (int n\_hor\_axis, int n\_vert\_axis, storey\_width, storey\_height, half\_pz\_height=np.array([ ]), origin=[0, 0], first\_hor\_axis=1, first\_vert\_axis=1, show\_plot=True)  
*Function that declares and initialises the grid nodes of a frame.*
- def `DefineFrameNodesAndElementsSteelShape` (int n\_hor\_axis, int n\_vert\_axis, storey\_width, storey\_height, list list\_col, list list\_beam, int geo\_trans\_ID, N\_G=np.array([ ]), t\_dp=np.array([ ]), L\_b\_col=np.array([ ]), L\_b\_beam=np.array([ ]), fix\_support=True, show\_plot=True, panel\_zone=True)  
*WIP (Work In Progress).*
- def `DefineSubassemblageNodes` (beam\_left\_L\_cl, beam\_right\_L\_cl, col\_top\_L\_cl, col\_bottom\_L\_cl, depth\_col, depth\_beam, boundary\_condition=True, show\_plot=True)  
*Function that declares and initialises the grid nodes of an interior subassemblage.*
- def `Initialize2DModel` (data\_dir="Results")  
*Function that initialise the project creating the 2D model with 3 DOF per node and set up a directory for the results.*

## 8.16 GeometryTemplate.py

[Go to the documentation of this file.](#)

```
00001 """
00002 Module with geometry templates (nodes and/or elements with associated fibers, material models, etc).
00003 Carmine Schipani, 2021
00004 """
00005
00006 # Import libraries
00007 from openseespy.opensees import *
00008 import matplotlib.pyplot as plt
00009 import numpy as np
00010 import os
00011 from copy import copy, deepcopy
00012 import openseespy.postprocessing.Get_Rendering as opsplt
00013 from OpenSeesPyAssistant.Section import *
00014 from OpenSeesPyAssistant.DataManagement import *
00015 from OpenSeesPyAssistant.ErrorHandling import *
00016 from OpenSeesPyAssistant.Units import *
00017 from OpenSeesPyAssistant.Constants import *
00018 from OpenSeesPyAssistant.Fibers import *
00019 from OpenSeesPyAssistant.Connections import *
00020 from OpenSeesPyAssistant.FunctionalFeatures import *
00021 from OpenSeesPyAssistant.MemberModel import *
00022 from OpenSeesPyAssistant.AnalysisAndPostProcessing import *
00023
00024
00025 def Initialize2DModel(data_dir = "Results"):
00026     """
00027     Function that initialise the project creating the 2D model with 3 DOF per node and set up a
    directory for the results.
00028
00029     @param data_dir (str, optional): Directory where the data will be stored.
00030     The function forces the user to define it just for good practice and consistency between
    projects.
00031     Defaults to "Results".
00032     """
00033     # Clear all
00034     wipe()
00035
00036     # Build model (2D - 3 DOF/node)
00037     model('basic', '-ndm', 2, '-ndf', 3)
00038
00039     # Main Results Folder
00040     if not os.path.exists(data_dir):
00041         os.makedirs(data_dir)
00042
00043
00044 def DefineFrameNodes(n_hor_axis: int, n_vert_axis: int, storey_width, storey_height, half_pz_height =
np.array([ ]),
00045 origin = [0, 0], first_hor_axis = 1, first_vert_axis = 1, show_plot = True):
00046     """
00047     Function that declares and initialises the grid nodes of a frame. Option to offset the grid node
of the panel zones
00048     with the master node of the panel zone being the grid one (top center one). The function can
be used multiple times
00049     to create more complex geometries.
00050
```

```

00051     @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
00052     @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
00053     @param storey_width (float): Width of the bays.
00054     @param storey_height (float): Height of the storeys.
00055     @param half_pz_height (np.ndarray, optional): Array of 1 dimension with half the height of the
panel zone for each floor.
00056     The first floor should be 0 (no panel zone in the support). Defaults to np.array([]), e.g. no
panel zone.
00057     @param origin (list, optional): List of two entry with the origin position. Defaults to [0, 0].
00058     @param first_hor_axis (int, optional): Number of the first pier. Defaults to 1.
00059     @param first_vert_axis (int, optional): Number of the first floor. Defaults to 1.
00060     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
Defaults to True.
00061
00062     @exception NegativeValue: n_hor_axis needs to be a positive integer.
00063     @exception NegativeValue: n_vert_axis needs to be a positive integer.
00064     @exception NegativeValue: storey_width needs to be positive.
00065     @exception NegativeValue: storey_height needs to be positive.
00066     @exception WrongDimension: origin has a dimension of 2.
00067     @exception NegativeValue: first_hor_axis needs to be a positive integer.
00068     @exception NegativeValue: first_vert_axis needs to be a positive integer.
00069     @exception WrongDimension: size of half_pz_height needs to be equal to n_vert_axis, if different
from 0.
00070
00071     @returns list: List with the nodes declared.
00072     """
00073     if n_hor_axis < 1: raise NegativeValue()
00074     if n_vert_axis < 1: raise NegativeValue()
00075     if storey_width < 0: raise NegativeValue()
00076     if storey_height < 0: raise NegativeValue()
00077     if len(origin) != 2: raise WrongDimension()
00078     if first_hor_axis < 1: raise NegativeValue()
00079     if first_vert_axis < 1: raise NegativeValue()
00080     if np.size(half_pz_height) != 0 and np.size(half_pz_height) != n_vert_axis: raise WrongDimension()
00081
00082     if np.size(half_pz_height) == 0: half_pz_height = np.zeros(n_vert_axis)
00083     node_array = []
00084     max_n_x = n_hor_axis + first_hor_axis - 1
00085     max_n_y = n_vert_axis + first_vert_axis - 1
00086     for xx in range(n_hor_axis):
00087         x_axis = xx + first_hor_axis
00088         for yy in range(n_vert_axis):
00089             y_axis = yy + first_vert_axis
00090             node_ID = GridIDConvention(x_axis, y_axis, max_n_x, max_n_y)
00091             node(node_ID, origin[0]+xx*storey_width, origin[1]+yy*storey_height+half_pz_height[yy])
00092             if y_axis == 1 and half_pz_height[yy] != 0: print("Warning: the nodes at the base have a
panel zone height")
00093             node_array.append(node_ID)
00094
00095     if show_plot:
00096         plot_nodes(node_array, "Frame geometry template with only nodes", True)
00097         plt.grid()
00098
00099     return node_array
00100
00101
00102 def DefineFrameNodesAndElementsSteelIShape(n_hor_axis: int, n_vert_axis: int, storey_width,
storey_height,
00103     list_col: list, list_beam: list, geo_trans_ID: int, N_G = np.array([]), t_dp = np.array([]),
L_b_col = np.array([]), L_b_beam = np.array([]),
00104     fix_support = True, show_plot = True, panel_zone = True):
00105     """
00106     WIP (Work In Progress). Function that declares and initialises the grid nodes of a frame and the
members using steel I shape SpringBasedElements.
00107     WARNING: Current limit of the geometry: n_hor_axis and n_vert_axis < 10; if exceeded, there are
problems with the IDs (ID limit is exceeded, ~2.2e9).
00108     WARNING: if the section of the columns change, the function does not account for the splacing.
Each column section is defined from floor to floor;
00109     if there is a change in the column section, it happens right after the panel zone (not realistic
but good enough for predesign).
00110     WIP: Solve ID limit for large building need implementations (for example the use of a different ID
convention or the use of the class IDGenerator).
00111
00112     @param n_hor_axis (int): Number of horizontal axis (or piers) for the grid of the frame.
00113     @param n_vert_axis (int): Number of vertical axis (or floors) for the grid of the frame.
00114     @param storey_width (float): Width of the bays.
00115     @param storey_height (float): Height of the storeys.
00116     @param list_col (list(SteelIShape)): List with the sections of the columns for every floor.
00117     @param list_beam (list(SteelIShape)): List with the sections of the beams for every bay.
00118     @param geo_trans_ID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
00119     @param N_G (np.ndarray, optional): Array of dimension 1 with the axial load for each column
(starting at floor 2). Defaults to np.array([]), e.g. 0.
00120     @param t_dp (np.ndarray, optional): Array of dimension 1 with the doubler plate thickness for each
bay's beam. Defaults to np.array([]), e.g. 0.
00121     @param L_b_col (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral
buckling length for each column. Defaults to np.array([]), e.g. -1.

```



```

00122     @param L_b_beam (np.ndarray, optional): Array of dimension 1 with the maxiaml unbraced lateral
00123     buckling length for each beam. Defaults to np.array([]), e.g. -1.
00123     @param fix_support (bool, optional): Option to fix the support of the frame. Defaults to True.
00124     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
00124     Defaults to True.
00125     @param panel_zone (bool, optional): Option to add the panel zones in the model. Defaults to True.
00126
00127     @exception WrongDimension: N_G dimension needs to be equal to n_vert_axis-1, if different from 0.
00128     @exception WrongDimension: t_dp dimension needs to be equal to n_vert_axis-1, if different from 0.
00129     @exception WrongDimension: L_b_col dimension needs to be equal to n_vert_axis-1, if different from
00130     0.
00130     @exception WrongDimension: L_b_beam dimension needs to be equal to n_hor_axis-1, if different from
00131     0.
00131     @exception WrongDimension: list_col dimension needs to be equal to n_vert_axis-1.
00132     @exception WrongDimension: list_beam dimension needs to be equal to n_vert_axis-1.
00133     @exception NegativeValue: geo_trans_ID needs to be a positive integer.
00134
00135     @returns List: List with the element objects in the frame.
00136     """
00137     panel_zone = True
00138     if np.size(N_G) == 0: N_G = np.zeros(n_vert_axis-1)
00139     if np.size(t_dp) == 0: t_dp = np.zeros(n_vert_axis-1)
00140     if np.size(L_b_col) == 0: L_b_col = np.ones(n_vert_axis-1) * (-1.0)
00141     if np.size(L_b_beam) == 0: L_b_beam = np.ones(n_hor_axis-1) * (-1.0)
00142
00143     if np.size(list_col) != n_vert_axis-1: raise WrongDimension()
00144     if np.size(list_beam) != n_vert_axis-1: raise WrongDimension()
00145     if np.size(N_G) != n_vert_axis-1: raise WrongDimension()
00146     if np.size(t_dp) != n_vert_axis-1: raise WrongDimension()
00147     if np.size(L_b_col) != n_vert_axis-1: raise WrongDimension()
00148     if np.size(L_b_beam) != n_hor_axis-1: raise WrongDimension()
00149     if geo_trans_ID < 1: raise NegativeValue()
00150
00151     half_pz_height = np.zeros(n_vert_axis)
00152     if panel_zone:
00153         for ii, beam in enumerate(list_beam):
00154             half_pz_height[ii+1] = beam.d/2
00155
00156     node_array = DefineFrameNodes(n_hor_axis, n_vert_axis, storey_width, storey_height,
00157     half_pz_height, [0, 0], 1, 1, False)
00158
00158     beam_column_pzspring = [[], [], []]
00159     for xx in range(n_hor_axis):
00160         for yy in range(n_vert_axis):
00161             node_ID = node_array[xx*n_vert_axis + yy]
00162             if yy != 0:
00163                 # Panel Zone
00164                 if half_pz_height[yy] == 0:
00165                     col_j_node_ID = node_ID
00166                     beam_j_node_ID = node_ID
00167                 else:
00168                     tmp_pz = PanelZoneSteelIShapeSkiadopoulos2021(node_ID, list_col[yy-1],
00169     list_beam[yy-1], geo_trans_ID, t_dp[yy-1])
00170                     tmp_pz.CreateMember()
00171                     col_j_node_ID = IDConvention(node_ID, 5, 1)
00172                     beam_j_node_ID = IDConvention(node_ID, 8, 1)
00173                     beam_column_pzspring[2].append(deepcopy(tmp_pz))
00174
00175                 # Column
00176                 col_i_node_ID = node_array[xx*n_vert_axis + yy - 1]
00177                 col_mat_i = OffsetNodeIDConvention(col_i_node_ID, "vertical", "i")
00178                 col_mat_j = OffsetNodeIDConvention(col_j_node_ID, "vertical", "j")
00179                 ele_ID = col_i_node_ID if panel_zone else -1
00180                 tmp_col = SpringBasedElementModifiedIMKSteelIShape(col_i_node_ID, col_j_node_ID,
00181     list_col[yy-1], geo_trans_ID,
00182                 col_mat_i, col_mat_j, N_G[yy-1], L_b=L_b_col[yy-1], ele_ID = ele_ID)
00183                 tmp_col.CreateMember()
00184                 beam_column_pzspring[1].append(deepcopy(tmp_col))
00185
00186             if xx != 0:
00187                 # Beam
00188                 if half_pz_height[yy] == 0:
00189                     beam_i_node_ID = node_array[(xx-1)*n_vert_axis + yy]
00190                 else:
00191                     beam_i_node_ID = IDConvention(node_array[(xx-1)*n_vert_axis + yy], 2, 1)
00192                     beam_mat_i = OffsetNodeIDConvention(beam_i_node_ID, "horizontal", "i")
00193                     beam_mat_j = OffsetNodeIDConvention(beam_j_node_ID, "horizontal", "j")
00194                     ele_ID = beam_i_node_ID if panel_zone else -1
00195                     tmp_beam = SpringBasedElementModifiedIMKSteelIShape(beam_i_node_ID,
00196     beam_j_node_ID, list_beam[yy-1], geo_trans_ID,
00197                     beam_mat_i, beam_mat_j, L_b=L_b_beam[xx-1], ele_ID = ele_ID)
00198                     tmp_beam.CreateMember()
00199                     beam_column_pzspring[0].append(deepcopy(tmp_beam))
00200             else:
00201                 if fix_support: RigidSupport(node_ID)

```

```

00201     if show_plot:
00202         opsplt.plot_model("nodes", "elements")
00203
00204     return beam_column_pzspring
00205
00206
00207 def DefineSubassemblageNodes(beam_left_L_cl, beam_right_L_cl, col_top_L_cl, col_bottom_L_cl,
    depth_col, depth_beam,
00208     boundary_condition = True, show_plot = True):
00209     """
00210     Function that declares and initialises the grid nodes of an interior subassemblage. The panel zone
    geometry is defined by the two arguments
00211         depth_col and depth_beam.
00212
00213     @param beam_left_L_cl (float): Centerline length of the left beam (excluding the panel zone).
00214     @param beam_right_L_cl (float): Centerline length of the right beam (excluding the panle zone).
00215     @param col_top_L_cl (float): Centerline length of the top column (excluding the panel zone).
00216     @param col_bottom_L_cl (float): Centerline length of the bottom column (excluding the panel zone).
00217     @param depth_col (float): Depth of the columns for the panel zone.
00218     @param depth_beam (float): Depth of the beams for the panel zone.
00219     @param boundary_condition (bool, optional): Option to set already the boundary condition (bottom
    column pinned, beams fix only y movement).
    Defaults to True.
00220
00221     @param show_plot (bool, optional): Option to show the plot of the nodes declared and initialised.
    Defaults to True.
00222
00223     @exception NegativeValue: beam_left_L_cl needs to be positive.
00224     @exception NegativeValue: beam_right_L_cl needs to be positive.
00225     @exception NegativeValue: col_top_L_cl needs to be positive.
00226     @exception NegativeValue: col_bottom_L_cl needs to be positive.
00227     @exception NegativeValue: depth_col needs to be positive.
00228     @exception NegativeValue: depth_beam needs to be positive.
00229
00230     @returns list: List with the nodes declared.
00231     """
00232     # origin is the bottom left corner
00233     if beam_left_L_cl < 0: raise NegativeValue()
00234     if beam_right_L_cl < 0: raise NegativeValue()
00235     if col_top_L_cl < 0: raise NegativeValue()
00236     if col_bottom_L_cl < 0: raise NegativeValue()
00237     if depth_col < 0: raise NegativeValue()
00238     if depth_beam < 0: raise NegativeValue()
00239
00240     node(12, 0.0, col_bottom_L_cl+depth_beam/2)
00241     node(21, beam_left_L_cl+depth_col/2, 0.0)
00242     node(22, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam)
00243     node(23, beam_left_L_cl+depth_col/2, col_bottom_L_cl+depth_beam+col_top_L_cl)
00244     node(32, beam_left_L_cl+depth_col+beam_right_L_cl, col_bottom_L_cl+depth_beam/2)
00245     node_array = [12, 21, 22, 23, 32]
00246
00247     if boundary_condition:
00248         fix(12, 0, 1, 0)
00249         fix(32, 0, 1, 0)
00250         fix(21, 1, 1, 0)
00251
00252     if show_plot:
00253         plot_nodes(node_array, "Subassemblage geometry template with only nodes", True)
00254         plt.grid()
00255
00256     return node_array
00257
00258
00259 # def DefineRCSSubassemblage():
00260 #     # WIP and experimental
00261
00262

```

## 8.17 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MaterialModels.py File Reference ↩

### Classes

- class [ConfMander1988Circ](#)

*Class that stores funcions and material properties of a RC circular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.*

- class [ConfMander1988CircRCCircShape](#)

- Class that is the children of [ConfMander1988Circ](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.
- class [ConfMander1988Rect](#)

Class that stores functions and material properties of a RC rectangular section with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
  - class [ConfMander1988RectRCRectShape](#)

Class that is the children of [ConfMander1988Rect](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.
  - class [GMP1970](#)

Class that stores functions and material properties of the vertical steel reinforcement bars with Giuffr , Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type used to model it is Steel02.
  - class [GMP1970RCRectShape](#)

Class that is the children of [GMP1970](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.
  - class [Gupta1999](#)

Class that stores functions and material properties of a steel double symmetric I-shape profile with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.
  - class [Gupta1999SteelShape](#)

Class that is the children of [Gupta1999](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.
  - class [MaterialModels](#)

Parent abstract class for the storage and manipulation of a material model's information (mechanical and geometrical parameters, etc) and initialisation in the model.
  - class [ModifiedIMK](#)

Class that stores functions and material properties of a steel double symmetric I-shape profile with modified Ibarra- $\leftrightarrow$  Medina-Krawinkler as the material model for the nonlinear springs and the OpenSeesPy command type used to model it is Bilin.
  - class [ModifiedIMKSteelShape](#)

Class that is the children of [ModifiedIMK](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.
  - class [Skiadopoulos2021](#)

Class that stores functions and material properties of a steel double symmetric I-shape profile with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command type used to model it is Hysteresis.
  - class [Skiadopoulos2021RCS](#)

WIP: Class that is the children of [Skiadopoulos2021](#) and it's used for the panel zone spring in a RCS (RC column continuous, Steel beam).
  - class [Skiadopoulos2021SteelShape](#)

Class that is the children of [Skiadopoulos2021](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.
  - class [UnconfMander1988](#)

Class that stores functions and material properties of a RC rectangular or circular section with Mander 1988 as the material model for the unconfined reinforced concrete and the OpenSeesPy command type used to model it is Concrete04 or Concrete01.
  - class [UnconfMander1988RCCircShape](#)

Class that is the children of [UnconfMander1988](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.
  - class [UnconfMander1988RCRectShape](#)

Class that is the children of [UnconfMander1988](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.
  - class [UniaxialBilinear](#)

Class that stores functions and material properties of a simple uniaxial bilinear model with the OpenSeesPy command type used to model it is Steel01.
  - class [UniaxialBilinearSteelShape](#)

Class that is the children of [UniaxialBilinear](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

- class [UVC](#)

Class that stores functions and material properties of a steel profile or reinforcing bar with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model it is [UVCuniaxial](#).

- class [UVCCalibrated](#)

Class that is the children of [UVC](#) that retrieve calibrated parameters from [UVC\\_calibrated\\_parameters.txt](#).

- class [UVCCalibratedRCCircShape](#)

Class that is the children of [UVCCalibrated](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.

- class [UVCCalibratedRCRectShape](#)

Class that is the children of [UVCCalibrated](#) and combines the class [RCRectShape](#) (section) to retrieve the information needed.

- class [UVCCalibratedSteelShapeFlange](#)

Class that is the children of [UVCCalibrated](#) and combine the class [SteelShape](#) (section) to retrieve the information needed for the material model of the flange (often used for the entire section).

- class [UVCCalibratedSteelShapeWeb](#)

Class that is the children of [UVCCalibrated](#) and combine the class [SteelShape](#) (section) to retrieve the information needed for the material model of the web.

## Namespaces

- namespace [MaterialModels](#)

Module for the material models.

## Functions

- def [Concrete01Funct](#) (fc, ec, fpcu, ecu, discretized\_eps)

Function with the equation of the curve of the [Concrete01](#) model.

- def [Concrete04Funct](#) (fc, discretized\_eps, ec, Ec)

Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).

- def [PlotConcrete01](#) (fc, ec, fpcu, ecu, ax, ID=0)

Function that plots the [Concrete01](#) stress-strain curve.

- def [PlotConcrete04](#) (fc, Ec, ec, ecu, str Type, ax, ID=0)

Function that plots the confined/unconfined [Concrete04](#) stress-strain curve.

## 8.18 MaterialModels.py

[Go to the documentation of this file.](#)

```
00001 """
00002 Module for the material models.
00003 Carmine Schipani, 2021
00004 """
00005
00006 from openseespy.opensees import *
00007 import matplotlib.pyplot as plt
00008 import numpy as np
00009 import os
00010 import math
00011 from abc import abstractmethod
00012 from copy import copy, deepcopy
00013 from OpenSeesPyAssistant.Section import *
00014 from OpenSeesPyAssistant.DataManagement import *
00015 from OpenSeesPyAssistant.ErrorHandling import *
00016 from OpenSeesPyAssistant.Units import *
00017
```

```

00018
00019 class MaterialModels(DataManagement):
00020     """
00021     Parent abstract class for the storage and manipulation of a material model's information
(mechanical
    and geometrical parameters, etc) and initialisation in the model.
00022
00023     @param DataManagement: Parent abstract class.
00024     """
00025     @abstractmethod
00026     def CheckApplicability(self):
00027         """
00028         Abstract function used to check the applicability of the material model.
00029         """
00030         pass
00031
00032
00033
00034 class ModifiedIMK(MaterialModels):
00035     """
00036     Class that stores functions and material properties of a steel double symmetric I-shape profile
00037     with modified Ibarra-Medina-Krawinkler as the material model for the nonlinear springs and the
OpenSeesPy command type used to model it is Bilin.
00038     The default values are valid for a simple cantilever.
00039     For more information about the empirical model for the computation of the parameters, see Lignos
Krawinkler 2011.
00040     The parameter 'n' is used as global throughout the SteelIShape sections to optimise the program
(given the fact that is constant everytime).
00041
00042     @param MaterialModels: Parent abstract class.
00043     """
00044     global n
00045     n = 10.0
00046
00047     def __init__(self, ID: int, Type: str, d, bf, tf, tw, h_1, Iy_mod, iz, E, Fy, Npl, My, L,
00048     N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K = -1, theta_u = -1, safety_factors =
False):
00049         """
00050         Constructor of the class. Every argument that is optional and is initialised as -1, will be
computed in this class.
00051
00052         @param ID (int): ID of the material model.
00053         @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
00054         @param d (float): Depth of the section.
00055         @param bf (float): Flange's width of the section
00056         @param tf (float): Flange's thickness of the section
00057         @param tw (float): Web's thickness of the section
00058         @param h_1 (float): Depth excluding the flange's thicknesses and the weld fillets.
00059         @param Iy_mod (float): n modified moment of inertia (strong axis)
00060         @param iz (float): Radius of gyration (weak axis).
00061         @param E (float): Young modulus.
00062         @param Fy (float): Yield strength.
00063         @param Npl (float): Maximal vertical axial load.
00064         @param My (float): Yielding moment.
00065         @param L (float): Effective length of the element associated with this section.
00066         If the panel zone is present, exclude its dimension.
00067         @param N_G (float, optional): Gravity axial load. Defaults to 0.
00068         @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
00069         @param L_0 (float, optional): Position of the inflection point.
00070         Defaults to -1, e.g. computed as the total length, assuming cantilever.
00071         @param L_b (float, optional): Maximal unbraced lateral torsional buckling length.
00072         Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing
support.
00073         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00074         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
ComputeK.
00075         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
ComputeTheta_u.
00076         @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters
are used (not test results). Defaults to False.
00077
00078         @exception NegativeValue: ID needs to be a positive integer.
00079         @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
00080         @exception NegativeValue: d needs to be positive.
00081         @exception NegativeValue: bf needs to be positive.
00082         @exception NegativeValue: tf needs to be positive.
00083         @exception NegativeValue: tw needs to be positive.
00084         @exception NegativeValue: h_1 needs to be positive.
00085         @exception NegativeValue: Iy_mod needs to be positive.
00086         @exception NegativeValue: iz needs to be positive.
00087         @exception NegativeValue: E needs to be positive.
00088         @exception NegativeValue: Fy needs to be positive.
00089         @exception NegativeValue: Npl needs to be positive.
00090         @exception NegativeValue: My needs to be positive.
00091         @exception NegativeValue: L needs to be positive.
00092         @exception NegativeValue: N_G needs to be positive.
00093         @exception NegativeValue: L_0 needs to be positive if different from -1.
00094         @exception NegativeValue: L_b needs to be positive if different from -1.

```

```

00095         @exception NegativeValue: Mc needs to be positive if different from -1.
00096         @exception NegativeValue: K needs to be positive if different from -1.
00097         @exception NegativeValue: theta_u needs to be positive if different from -1.
00098         @exception InconsistentGeometry: h_l can't be bigger than d
00099         @exception MemberFailure: N_G can't be bigger than Npl (section failure).
00100         @exception InconsistentGeometry: L_0 can't be bigger than L
00101         """
00102         # Check
00103         if ID < 0: raise NegativeValue()
00104         if Type != "Beam" and Type != "Col": raise WrongArgument()
00105         if d < 0: raise NegativeValue()
00106         if bf < 0: raise NegativeValue()
00107         if tf < 0: raise NegativeValue()
00108         if tw < 0: raise NegativeValue()
00109         if h_l < 0: raise NegativeValue()
00110         if Iy_mod < 0: raise NegativeValue()
00111         if iz < 0: raise NegativeValue()
00112         if E < 0: raise NegativeValue()
00113         if Fy < 0: raise NegativeValue()
00114         if Npl < 0: raise NegativeValue()
00115         if My < 0: raise NegativeValue()
00116         if L < 0: raise NegativeValue()
00117         if N_G < 0: raise NegativeValue()
00118         if L_0 != -1 and L_0 < 0: raise NegativeValue()
00119         if L_b != -1 and L_b < 0: raise NegativeValue()
00120         if Mc != -1 and Mc < 0: raise NegativeValue()
00121         if K != -1 and K < 0: raise NegativeValue()
00122         if theta_u != -1 and theta_u < 0: raise NegativeValue()
00123         if h_l > d: raise InconsistentGeometry()
00124         if N_G > Npl: raise MemberFailure()
00125         if L_0 > L: raise InconsistentGeometry()
00126
00127         # Arguments
00128         self.TypeType = Type
00129         self.IDID = ID
00130         self.dd = d
00131         self.bfbf = bf
00132         self.tftf = tf
00133         self.twtw = tw
00134         self.h_lh_l = h_l
00135         self.Iy_modIy_mod = Iy_mod
00136         self.iziz = iz
00137         self.EE = E
00138         self.FyFy = Fy
00139         self.NplNpl = Npl
00140         self.MyMy = My
00141         self.LL = L
00142         self.N_GN_G = N_G
00143         self.K_factorK_factor = K_factor
00144         self.L_0L_0 = L if L_0 == -1 else L_0
00145         self.L_bL_b = L if L_b == -1 else L_b
00146
00147         # Initialized the parameters that are dependent from others
00148         self.section_name_tagsection_name_tag = "None"
00149         self.InitializedInitialized = False
00150         if safety_factors:
00151             self.gamma_rmgamma_rm = 1.25
00152             self.prob_factorprob_factor = 1.15
00153         else:
00154             self.gamma_rmgamma_rm = 1.0
00155             self.prob_factorprob_factor = 1.0
00156         self.ReInitReInit(Mc, K, theta_u)
00157
00158         # Methods
00159         def ReInit(self, Mc = -1, K = -1, theta_u = -1):
00160             """
00161             Implementation of the homonym abstract method.
00162             See parent class DataManagement for detailed information.
00163
00164             @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00165             @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
00166             ComputeK.
00167             @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
00168             ComputeTheta_u.
00169             """
00170             # Precompute some members
00171             self.My_starMy_star = self.ComputeMyStarComputeMyStar()
00172
00173             # Arguments
00174             self.McMc = self.ComputeMcComputeMc() if Mc == -1 else Mc
00175             self.KK = self.ComputeKComputeK() if K == -1 else K
00176             self.theta_utheta_u = self.ComputeTheta_uComputeTheta_u() if theta_u == -1 else theta_u
00177
00178             # Check applicability
00179             self.CheckApplicabilityCheckApplicabilityCheckApplicability()

```

```

00180         # Members
00181         self.KeKe = self.ComputeKeComputeKe()
00182         self.theta_ytheta_y = self.ComputeTheta_yComputeTheta_y()
00183         self.theta_ptheta_p = self.ComputeTheta_pComputeTheta_p()
00184         self.theta_pcttheta_pc = self.ComputeTheta_pcComputeTheta_pc()
00185         self.McMyMcMy = self.McMc/self.My_starMy_star
00186         self.rate_detrate_det = self.ComputeRefEnergyDissipationCapComputeRefEnergyDissipationCap()
00187         self.aa = self.ComputeaaComputeaa()
00188         self.a_sa_s = self.Computea_sComputea_s()
00189         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"

00190
00191         # Data storage for loading/saving
00192         self.UpdateStoredDataUpdateStoredData()
00193
00194
00195     def UpdateStoredData(self):
00196         """
00197         Implementation of the homonym abstract method.
00198         See parent class DataManagement for detailed information.
00199         """
00200         self.data = [{"INFO_TYPE": "ModifiedIMK"}, # Tag for differentiating different data
00201             ["ID", self.IDID],
00202             ["section_name_tag", self.section_name_tagsection_name_tag],
00203             ["Type", self.TypeType],
00204             ["d", self.dd],
00205             ["bf", self.bfbf],
00206             ["tf", self.tftf],
00207             ["tw", self.twtw],
00208             ["h_l", self.h_lh_l],
00209             ["Iy_mod", self.Iy_modIy_mod],
00210             ["iz", self.iziz],
00211             ["E", self.EE],
00212             ["Fy", self.FyFy],
00213             ["L", self.LL],
00214             ["N_G", self.N_GN_G],
00215             ["K_factor", self.K_factorK_factor],
00216             ["Ke", self.KeKe],
00217             ["L_0", self.L_0L_0],
00218             ["L_b", self.L_bL_b],
00219             ["gamma_rm", self.gamma_rmgamma_rm],
00220             ["prob_factor", self.prob_factorprob_factor],
00221             ["Npl", self.NplNpl],
00222             ["My", self.MyMy],
00223             ["My_star", self.My_starMy_star],
00224             ["Mc", self.McMc],
00225             ["McMy", self.McMyMcMy],
00226             ["K", self.KK],
00227             ["theta_y", self.theta_ytheta_y],
00228             ["theta_p", self.theta_ptheta_p],
00229             ["theta_pc", self.theta_pcttheta_pc],
00230             ["theta_u", self.theta_utheta_u],
00231             ["rate_det", self.rate_detrate_det],
00232             ["a", self.aa],
00233             ["a_s", self.a_sa_s],
00234             ["Initialized", self.InitializedInitialized]]
00235
00236
00237     def ShowInfo(self, plot = False, block = False):
00238         """
00239         Implementation of the homonym abstract method.
00240         See parent class DataManagement for detailed information.
00241
00242         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00243         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
00244         """
00245         Mr = self.KK*self.My_starMy_star
00246         theta_p_plot = self.theta_ptheta_p
00247         if self.theta_ptheta_p > self.theta_utheta_u-self.theta_ytheta_y:
00248             theta_p_plot = self.theta_utheta_u-self.theta_ytheta_y
00249         theta_r = self.theta_ytheta_y + theta_p_plot + self.theta_pcttheta_pc*(1.0-Mr/self.McMc)
00250         if theta_r > self.theta_utheta_u:
00251             theta_r = self.theta_utheta_u
00252         Mr =
self.McMc*(1.0-1.0/self.theta_pcttheta_pc*(self.theta_utheta_u-self.theta_ytheta_y-theta_p_plot))
00253
00254         print("")
00255         print("Requested info for Modified IMK (Ibarra-Medina-Krawinkler) material model Parameters,
ID = {}".format(self.IDID))
00256         print("Section associated: {}".format(self.section_name_tagsection_name_tag))
00257         print('theta y = {}'.format(self.theta_ytheta_y))
00258         print('theta p = {}'.format(self.theta_ptheta_p))
00259         print('theta r = {}'.format(theta_r))
00260         print('theta pc = {}'.format(self.theta_pcttheta_pc))
00261         print('theta u = {}'.format(self.theta_utheta_u))

```

```

00262         print('My star = {} kNm'.format(self.My_starMy_star/kNm_unit))
00263         print('Mc = {} kNm'.format(self.McMc/kNm_unit))
00264         print('Mr = {} kNm'.format(Mr/kNm_unit))
00265         print('a = {} '.format(self.aa))
00266         print('as = {} '.format(self.a_sa_s))
00267         print('lambda (deterioration rate) = {} '.format(self.rate_detrate_det))
00268         print("")
00269
00270         if plot:
00271             # Data for plotting
00272             x_axis = np.array([0.0, self.theta_ytheta_y, self.theta_ytheta_y + theta_p_plot, theta_r,
self.theta_utheta_u, self.theta_utheta_u])
00273             x_axis2 = np.array([self.theta_ytheta_y + theta_p_plot, self.theta_ytheta_y + theta_p_plot
+ self.theta_pctheta_pc])
00274             y_axis = np.array([0.0, self.My_starMy_star, self.McMc, Mr, Mr, 0.0])/kNm_unit
00275             y_axis2 = np.array([self.McMc, 0.0])/kNm_unit
00276
00277             fig, ax = plt.subplots()
00278             ax.plot(x_axis, y_axis, 'k-')
00279             ax.plot(x_axis2, y_axis2, 'k--')
00280
00281             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
00282                   title='Modified IMK deterioration model (ID={})'.format(self.IDID))
00283             ax.grid()
00284
00285             if block:
00286                 plt.show()
00287
00288
00289     def CheckApplicability(self):
00290         """
00291         Implementation of the homonym abstract method.
00292         See parent class MaterialModels for detailed information.
00293         """
00294         Check = True
00295         if self.TypeType == "Beam":
00296             if self.dd/self.twtw < 20 or self.dd/self.twtw > 55:
00297                 Check = False
00298                 print("The d/tw check was not fullfilled")
00299             if self.L_bL_b/self.iziz < 20 or self.L_bL_b/self.iziz > 80:
00300                 Check = False
00301                 print("The Lb/iz check was not fullfilled")
00302             if self.bfbf/2/self.tftf < 4 or self.bfbf/2/self.tftf > 8:
00303                 Check = False
00304                 print("The bf/2/tf check was not fullfilled")
00305             if self.LL/self.dd < 2.5 or self.LL/self.dd > 7:
00306                 Check = False
00307                 print("The check L/d was not fullfilled")
00308             if self.dd < 102*mm_unit or self.dd > 914*mm_unit:
00309                 Check = False
00310                 print("The d check was not fullfilled")
00311             if self.FyFy < 240*MPa_unit or self.FyFy > 450*MPa_unit:
00312                 Check = False
00313                 print("The Fy check was not fullfilled")
00314         else:
00315             if self.h_lh_l/self.twtw < 3.71 or self.dd/self.twtw > 57.5:
00316                 Check = False
00317                 print("The hl/tw check was not fullfilled")
00318             if self.L_bL_b/self.iziz < 38.4 or self.L_bL_b/self.iziz > 120:
00319                 Check = False
00320                 print("The Lb/iz check was not fullfilled")
00321             if self.N_GN_G/self.NplNpl < 0 or self.N_GN_G/self.NplNpl > 0.75:
00322                 Check = False
00323                 print("The NG/Npl check was not fullfilled")
00324         if not Check:
00325             print("The validity of the equations is not fullfilled.")
00326             print("!!!!!! WARNING !!!!!!! Check material model of Modified IMK, ID=", self.IDID)
00327             print("")
00328
00329     def ComputeKe(self):
00330         """
00331         Method that computes the elastic stiffness.
00332
00333         @returns float: The stiffness
00334         """
00335         return self.K_factorK_factor*n*self.EE*self.Iy_modIy_mod/self.LL
00336
00337     def Computea(self):
00338         """
00339         Method that computes the strain hardening ratio with the n modification.
00340
00341         @returns float: Strain hardening ratio.
00342         """
00343         # strain hardening ratio of spring
00344         return (n+1.0)*self.My_starMy_star*(self.McMyMcMy-1.0)/(self.KeKe*self.theta_ptheta_p)
00345
00346     def Computea_s(self):

```



```

00347         """
00348         Method that computes the modified strain hardening ratio for the spring.
00349         For more info see Ibarra & Krawinkler 2005.
00350
00351         @returns float: Strain hardening ratio.
00352         """
00353         return self.aa/(1.0+n*(1.0-self.aa))
00354
00355     def ComputeMyStar(self):
00356         """
00357         Method that computes the effective yield moment.
00358         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00359
00360         @returns float: Effective yield moment.
00361         """
00362         if self.TypeType == "Beam":
00363             return self.prob_factorprob_factor*self.MyMy*self.gamma_rmgamma_rm*1.1
00364         else:
00365             if self.N_GN_G/self.NplNpl > 0.2:
00366                 return
00367             1.15*self.prob_factorprob_factor*self.MyMy*self.gamma_rmgamma_rm*(1-self.N_GN_G/self.NplNpl)*9.0/8.0
00368             else:
00369                 return
00370             1.15*self.prob_factorprob_factor*self.MyMy*self.gamma_rmgamma_rm*(1-self.N_GN_G/2.0/self.NplNpl)
00371
00372     def ComputeMc(self):
00373         """
00374         Method that computes the capping moment.
00375         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00376
00377         @returns float: Capping moment.
00378         """
00379         if self.TypeType == "Beam":
00380             return self.My_starMy_star*1.11
00381             # For RBS: My_star*1.09
00382         else:
00383             tmp =
00384             12.5*(self.h_lh_l/self.twtw)**(-0.2)*(self.L_bL_b/self.iziz)**(-0.4)*(1-self.N_GN_G/self.NplNpl)**0.4
00385             return max(min(1.3, tmp), 1.0)*self.My_starMy_star
00386
00387     def ComputeK(self):
00388         """
00389         Method that computes the residual strength ratio.
00390         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00391
00392         @returns float: Residual strength ratio.
00393         """
00394         if self.TypeType == "Beam":
00395             return 0.4
00396         else:
00397             tmp = 0.5-0.4*self.N_GN_G/self.NplNpl
00398             return max(tmp, 0)
00399
00400     def ComputeTheta_y(self):
00401         """
00402         Method that computes the yield rotation.
00403         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00404
00405         @returns float: Yield rotation.
00406         """
00407         return self.My_starMy_star/self.KeKe*(n+1)
00408
00409     def ComputeTheta_p(self):
00410         """
00411         Method that computes the plastic rotation.
00412         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00413
00414         @returns float: Plastic rotation.
00415         """
00416         if self.TypeType == "Beam":
00417             if self.dd < 533.0*mm_unit:
00418                 return
00419                 0.0865*(self.h_lh_l/self.twtw)**(-0.365)*(self.bfbf/2.0/self.tftf)**(-0.14)*(self.L_0L_0/self.dd)**(0.34)*(self.dd/(533.0*mm_unit))
00420             else:
00421                 return
00422                 0.318*(self.h_lh_l/self.twtw)**(-0.550)*(self.bfbf/2.0/self.tftf)**(-0.345)*(self.L_0L_0/self.dd)**(0.090)*(self.L_bL_b/self.dd)**(0.010)
00423             # With RBS: ...
00424         else:
00425             tmp =
00426             294.0*(self.h_lh_l/self.twtw)**(-1.7)*(self.L_bL_b/self.iziz)**(-0.7)*(1-self.N_GN_G/self.NplNpl)**(1.6)
00427             # *(self.E/self.Fy/gamma_rm)**(0.2) # EC8
00428             if tmp > 0.2:
00429                 tmp = 0.2
00430             # if tmp > self.theta_u-self.theta_y:
00431             #     tmp = (self.theta_u-self.theta_y)*0.799 # convergence issue
00432             return tmp

```

```

00427     def ComputeTheta_pc(self):
00428         """
00429         Method that computes the post capping rotation.
00430         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00431
00432         @returns float: Post capping rotation.
00433         """
00434         if self.TypeType == "Beam":
00435             if self.dd < 533.0*mm_unit:
00436                 return
00437             5.63*(self.h_1h_1/self.twtw)**(-0.565)*(self.bfbf/2.0/self.tftf)**(-0.800)*(self.dd/(533.0*mm_unit))**(-0.280)*(self.Fy/
00438             else:
00439                 return
00440             7.50*(self.h_1h_1/self.twtw)**(-0.610)*(self.bfbf/2.0/self.tftf)**(-0.710)*(self.L_bL_b/self.iziz)**(-0.110)*(self.dd/
00441             # With RBS: ...
00442         else:
00443             tmp =
00444             90.0*(self.h_1h_1/self.twtw)**(-0.8)*(self.L_bL_b/self.iziz)**(-0.8)*(1.0-self.N_GN_G/self.NplNpl)**(2.5)
00445             # *(self.E/self.Fy/gamma_rm)**(0.07) # EC8
00446             return min(tmp, 0.3)
00447
00448     def ComputeTheta_u(self):
00449         """
00450         Method that computes the ultimate rotation.
00451         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00452
00453         @returns float: Ultimate rotation.
00454         """
00455         if self.TypeType == "Beam":
00456             return 0.2
00457         else:
00458             return 0.15
00459
00460     def ComputeRefEnergyDissipationCap(self):
00461         """
00462         Method that computes the reference energy dissipation capacity.
00463         For more info see Lignos & Krawinkler 2011 and Lignos et Al. 2019.
00464
00465         @returns float: Reference energy dissipation capacity.
00466         """
00467         if self.TypeType == "Beam":
00468             if self.dd < 533.0*mm_unit:
00469                 return
00470             495.0*(self.h_1h_1/self.twtw)**(-1.34)*(self.bfbf/2.0/self.tftf)**(-0.595)*(self.FyFy/(355.0*MPa_unit))**(-0.360)
00471             else:
00472                 return
00473             536.0*(self.h_1h_1/self.twtw)**(-1.26)*(self.bfbf/2.0/self.tftf)**(-0.525)*(self.L_bL_b/self.iziz)**(-0.130)*(self.FyFy/
00474             # With RBS: ...
00475         else:
00476             if self.N_GN_G/self.NplNpl > 0.35:
00477                 tmp =
00478                 268000.0*(self.h_1h_1/self.twtw)**(-2.30)*(self.L_bL_b/self.iziz)**(-1.130)*(1.0-self.N_GN_G/self.NplNpl)**(1.19)
00479                 return min(tmp, 3.0)
00480             else:
00481                 tmp =
00482                 25000.0*(self.h_1h_1/self.twtw)**(-2.14)*(self.L_bL_b/self.iziz)**(-0.53)*(1.0-self.N_GN_G/self.NplNpl)**(4.92)
00483                 return min(tmp, 3.0)
00484
00485     def Bilin(self):
00486         """
00487         Generate the material model Bilin (Modified IMK) using the computed parameters.
00488         See _Bilin function for more information.
00489         """
00490         _Bilin(self.IDID, self.KeKe, self.a_sa_s, self.My_starMy_star, self.theta_ptheta_p,
00491         self.theta_ptheta_pc, self.KK, self.theta_utheta_u, self.rate_detrate_det)
00492         self.InitializedInitialized = True
00493         self.UpdateStoredDataUpdateStoredData()
00494
00495     class ModifiedIMKSteelIShape(ModifiedIMK):
00496         """
00497         Class that is the children of ModifiedIMK and combine the class SteelIShape (section) to retrieve
00498         the information needed.
00499
00500         @param ModifiedIMK: Parent class.
00501         """
00502         def __init__(self, ID, section: SteelIShape, N_G = 0, K_factor = 3, L_0 = -1, L_b = -1, Mc = -1, K
00503         = -1, theta_u = -1, safety_factors = False):
00504             """
00505             Constructor of the class. It passes the arguments into the parent class to generate the
00506             combination of the parent class
00507             and the section class SteelIShape.
00508             Every argument that is optional and is initialised as -1, will be computed in this class.
00509             The copy of the section passed is stored in the member variable self.sectionsection.
00510
00511             @param ID (int): ID of the material model.

```

```

00502         @param section (SteelIShape): Object that store informations for a steel I shpae section.
00503         @param N_G (float, optional): Gravity axial load. Defaults to 0.
00504         @param K_factor (float, optional): Rigidity factor. Defaults to 3 (assuming cantilever).
00505         @param L_0 (float, optional): Position of the inflection point.
00506             Defaults to -1, e.g. computed as the total length, assuming cantilever.
00507         @param L_b (float, optional):Maximal unbraced lateral torsional buckling length.
00508             Defaults to -1, e.g. computed as the total length, assuming cantilever with no bracing
support.
00509         @param Mc (float, optional): Capping moment. Defaults to -1, e.g. computed in ComputeMc.
00510         @param K (float, optional): Residual strength ratio. Defaults to -1, e.g. computed in
ComputeK.
00511         @param theta_u (float, optional): Ultimate rotation. Defaults to -1, e.g. computed in
ComputeTheta_u.
00512         @param safety_factors (bool, optional): Safety factors used if standard mechanical parameters
are used (not test results). Defaults to False.
00513         """
00514         self.sectionsection = deepcopy(section)
00515         super().__init__(ID, section.Type, section.d, section.bf, section.tf, section.tw, section.h_1,
00516             section.Iy_mod, section.iz, section.E, section.Fy, section.Npl, section.My, section.L,
N_G,
00517             K_factor, L_0, L_b, Mc, K, theta_u, safety_factors)
00518         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
00519         self.UpdateStoredDataUpdateStoredData()
00520
00521
00522 class Gupta1999(MaterialModels):
00523     """
00524     Class that stores functions and material properties of a steel double symmetric I-shape profile
00525     with Gupta 1999 as the material model for the panel zone and the OpenSeesPy command type used
to model it is Hysteresis.
00526     The material model is valid only if the column is continuous.
00527     For more information about the empirical model for the computation of the parameters, see Gupta
1999.
00528
00529     @param MaterialModels: Parent abstract class.
00530     """
00531     def __init__(self, ID: int, d_c, bf_c, tf_c, I_c, d_b, tf_b, Fy, E, t_p,
00532         t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
safety_factor = False):
00533         """
00534         Constructor of the class.
00535
00536         @param ID (int): Unique material model ID.
00537         @param d_c (float): Column depth.
00538         @param bf_c (float): Column flange width.
00539         @param tf_c (float): Column flange thickness.
00540         @param I_c (float): Column moment of inertia (strong axis).
00541         @param d_b (float): Beam depth.
00542         @param tf_b (float): Beam flange thickness.
00543         @param Fy (float): Yield strength (if assume continous column, Fy of the web).
00544         @param E (float): Young modulus.
00545         @param t_p (float): Panel zone thickness.
00546         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00547         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00548         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
Defaults to 0.25.
00549         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
Defaults to 0.75.
00550         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00551         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00552         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
on ductility, mu-beta. Defaults to 0.0.
00553         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
are used (not test results). Defaults to False.
00554
00555         @exception NegativeValue: ID needs to be a positive integer.
00556         @exception NegativeValue: d_c needs to be positive.
00557         @exception NegativeValue: bf_c needs to be positive.
00558         @exception NegativeValue: tf_c needs to be positive.
00559         @exception NegativeValue: d_b needs to be positive.
00560         @exception NegativeValue: tf_b needs to be positive.
00561         @exception NegativeValue: Fy needs to be positive.
00562         @exception NegativeValue: E needs to be positive.
00563         @exception NegativeValue: t_p needs to be positive.
00564         @exception NegativeValue: a_s needs to be positive.
00565         """
00566         # Check
00567         if ID < 1: raise NegativeValue()
00568         if d_c < 0: raise NegativeValue()
00569         if bf_c < 0: raise NegativeValue()
00570         if tf_c < 0: raise NegativeValue()
00571         if d_b < 0: raise NegativeValue()
00572         if tf_b < 0: raise NegativeValue()
00573         if Fy < 0: raise NegativeValue()
00574         if E < 0: raise NegativeValue()
00575         if t_p < 0: raise NegativeValue()
00576         if a_s < 0: raise NegativeValue()

```

```

00577
00578     # Arguments
00579     self.IDID = ID
00580     self.d_cd_c = d_c
00581     self.bf_cbfc_c = bf_c
00582     self.tf_ctfc_c = tf_c
00583     self.I_cI_c = I_c
00584     self.d_bd_b = d_b
00585     self.tf_btfc_b = tf_b
00586     self.FyFy = Fy
00587     self.EE = E
00588     self.t_pt_p = t_p
00589     self.t_dpt_dp = t_dp
00590     self.a_sa_s = a_s
00591     self.pinchxpinchx = pinchx
00592     self.pinchypinchy = pinchy
00593     self.dmg1dmg1 = dmg1
00594     self.dmg2dmg2 = dmg2
00595     self.betabeta = beta
00596     if safety_factor:
00597         self.RyRy = 1.2
00598     else:
00599         self.RyRy = 1.0
00600
00601     # Initialized the parameters that are dependent from others
00602     self.beam_section_name_tagbeam_section_name_tag = "None"
00603     self.col_section_name_tagcol_section_name_tag = "None"
00604     self.InitializedInitialized = False
00605     self.ReinitReinit()
00606
00607
00608     # Methods
00609     def ReInit(self):
00610         """
00611         Implementation of the homonym abstract method.
00612         See parent class DataManagement for detailed information.
00613         """
00614         # Check applicability
00615         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
00616
00617         # Members
00618         if self.beam_section_name_tagbeam_section_name_tag != "None":
00619             self.beam_section_name_tagbeam_section_name_tag = self.beam_section_name_tagbeam_section_name_tag + "
(modified)"
00620         if self.col_section_name_tagcol_section_name_tag != "None":
00621             self.col_section_name_tagcol_section_name_tag = self.col_section_name_tagcol_section_name_tag + "
(modified)"
00622
00623     # Trilinear Parameters
00624     self.t_pzt_pz = self.t_pt_p + self.t_dpt_dp
00625     self.VyVy = 0.55 * self.FyFy * self.RyRy * self.d_cd_c * self.t_pzt_pz # Yield Shear
00626     self.GG = self.EE / (2.0 * (1.0 + 0.30)) # Shear Modulus
00627     self.KeKe = 0.95 * self.GG * self.t_pzt_pz * self.d_cd_c # Elastic Stiffness
00628     self.KpKp = 0.95 * self.GG * self.bf_cbfc_c * (self.tf_ctfc_c * self.tf_ctfc_c) / self.d_bd_b #
Plastic Stiffness
00629
00630     # Define Trilinear Equivalent Rotational Spring
00631     # Yield point for Trilinear Spring at gamma1_y
00632     self.gammal_ygamma1_y = self.VyVy / self.KeKe
00633     self.M1yM1y = self.gammal_ygamma1_y * (self.KeKe * self.d_bd_b)
00634     # Second Point for Trilinear Spring at 4 * gamma1_y
00635     self.gamma2_ygamma2_y = 4.0 * self.gammal_ygamma1_y
00636     self.M2yM2y = self.M1yM1y + (self.KpKp * self.d_bd_b) * (self.gamma2_ygamma2_y -
self.gammal_ygamma1_y)
00637     # Third Point for Trilinear Spring at 100 * gamma1_y
00638     self.gamma3_ygamma3_y = 100.0 * self.gammal_ygamma1_y
00639     self.M3yM3y = self.M2yM2y + (self.a_sa_s * self.KeKe * self.d_bd_b) * (self.gamma3_ygamma3_y -
self.gamma2_ygamma2_y)
00640
00641     # Data storage for loading/saving
00642     self.UpdateStoredDataUpdateStoredData()
00643
00644     def UpdateStoredData(self):
00645         """
00646         Implementation of the homonym abstract method.
00647         See parent class DataManagement for detailed information.
00648         """
00649         self.datadata = [{"INFO_TYPE", "Gupta1999"}, # Tag for differentiating different data
[ "ID", self.IDID],
[ "beam_section_name_tag", self.beam_section_name_tagbeam_section_name_tag],
[ "col_section_name_tag", self.col_section_name_tagcol_section_name_tag],
[ "d_c", self.d_cd_c],
[ "bf_c", self.bf_cbfc_c],
[ "tf_c", self.tf_ctfc_c],
[ "I_c", self.I_cI_c],
[ "d_b", self.d_bd_b],

```

```

00657         ["tf_b", self.tf_bt_b],
00658         ["Fy", self.FyFy],
00659         ["E", self.EE],
00660         ["G", self.GG],
00661         ["t_p", self.t_pt_p],
00662         ["t_dp", self.t_dpt_dp],
00663         ["t_pz", self.t_pzt_pz],
00664         ["a_s", self.a_sa_s],
00665         ["pinchx", self.pinchxpinchx],
00666         ["pinchy", self.pinchypinchy],
00667         ["dmg1", self.dmg1dmg1],
00668         ["dmg2", self.dmg2dmg2],
00669         ["beta", self.betabeta],
00670         ["Ry", self.RyRy],
00671         ["Vy", self.VyVy],
00672         ["Ke", self.KeKe],
00673         ["Kp", self.KpKp],
00674         ["gamma1_y", self.gamma1_ygamma1_y],
00675         ["M1y", self.M1yM1y],
00676         ["gamma2_y", self.gamma2_ygamma2_y],
00677         ["M2y", self.M2yM2y],
00678         ["gamma3_y", self.gamma3_ygamma3_y],
00679         ["M3y", self.M3yM3y],
00680         ["Initialized", self.InitializedInitialized]]
00681
00682     def ShowInfo(self, plot = False, block = False):
00683         """
00684         Implementation of the homonym abstract method.
00685         See parent class DataManagement for detailed information.
00686
00687         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
00688         False.
00689         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
00690         of the program everytime that a plot should pop up). Defaults to False.
00691         """
00692         print("")
00693         print("Requested info for Gupta 1999 material model Parameters, ID = {}".format(self.IDID))
00694         print("Sections associated, column: {}".format(self.col_section_name_tagcol_section_name_tag))
00695         print("Sections associated, beam: {}".format(self.beam_section_name_tagbeam_section_name_tag))
00696         print("gamma1_y = {} rad".format(self.gamma1_ygamma1_y))
00697         print("gamma2_y = {} rad".format(self.gamma2_ygamma2_y))
00698         print("gamma3_y = {} rad".format(self.gamma3_ygamma3_y))
00699         print("M1y = {} kNm".format(self.M1yM1y/kNm_unit))
00700         print("M2y = {} kNm".format(self.M2yM2y/kNm_unit))
00701         print("M3y = {} kNm".format(self.M3yM3y/kNm_unit))
00702         print("")
00703         if plot:
00704             # Data for plotting
00705             # Last point for plot
00706             gamma3_y_plot = 10.0 * self.gamma3_ygamma3_y
00707             M3y_plot = self.M2yM2y + (self.a_sa_s * self.KeKe * self.d_bd_b) * (gamma3_y_plot -
00708             self.gamma2_ygamma2_y)
00709
00710             x_axis = np.array([0.0, self.gamma1_ygamma1_y, self.gamma2_ygamma2_y, gamma3_y_plot])
00711             y_axis = np.array([0.0, self.M1yM1y, self.M2yM2y, M3y_plot])/kNm_unit
00712
00713             fig, ax = plt.subplots()
00714             ax.plot(x_axis, y_axis, 'k-')
00715
00716             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
00717                   title='Gupta 1999 material model (ID={})'.format(self.IDID))
00718             ax.grid()
00719
00720             if block:
00721                 plt.show()
00722
00723     def CheckApplicability(self):
00724         """
00725         Implementation of the homonym abstract method.
00726         See parent class MaterialModels for detailed information.
00727         """
00728         Check = True
00729         # No checks
00730         if not Check:
00731             print("The validity of the equations is not fullfilled.")
00732             print("!!!!!! WARNING !!!!!!! Check material model of Gupta 1999, ID=", self.IDID)
00733             print("")
00734
00735     def Hysteretic(self):
00736         """
00737         Generate the material model Hysteretic (Gupta 1999) using the computed parameters.
00738         See _Hysteretic function for more information.

```

```

00739         """
00740         _Hysteretic(self.IDID, self.M1yM1y, self.gammal1_ygammal1_y, self.M2yM2y, self.gamma2_ygamma2_y,
self.M3yM3y, self.gamma3_ygamma3_y,
00741             self.pinchxpinchx, self.pinchypinchy, self.dmg1dmg1, self.dmg2dmg2, self.betabeta)
00742         self.InitializedInitialized = True
00743         self.UpdateStoredDataUpdateStoredData()
00744
00745
00746 class Gupta1999SteelIShape(Gupta1999):
00747     """
00748     Class that is the children of Gupta1999 and combine the class SteelIShape (section) to retrieve
the information needed.
00749
00750     @param Gupta1999: Parent class.
00751     """
00752     def __init__(self, ID: int, col: SteelIShape, beam: SteelIShape,
00753         t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
safety_factor = False):
00754         """
00755         Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
00756             and the section class SteelIShape.
00757         The copy of the sections (col and beam) passed is stored in the member variable self.section.
00758
00759         @param ID (int): Unique material model ID.
00760         @param col (SteelIShape): SteelIShape column section object.
00761         @param beam (SteelIShape): SteelIShape beam section object.
00762         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00763         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00764         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
Defaults to 0.25.
00765         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
Defaults to 0.75
00766         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00767         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00768         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
on ductility, mu-beta. Defaults to 0.0.
00769         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
are used (not test results). Defaults to False.
00770         """
00771         self.colcol = deepcopy(col)
00772         self.beambeam = deepcopy(beam)
00773         super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E,
col.tw,
00774             t_dp, a_s, pinchx, pinchy, dmg1, dmg2, beta, safety_factor)
00775         self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
00776         self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
00777         self.UpdateStoredDataUpdateStoredData()
00778
00779
00780 class Skiadopoulos2021(MaterialModels):
00781     """
00782     Class that stores functions and material properties of a steel double symmetric I-shape profile
with Skiadopoulos 2021 as the material model for the panel zone and the OpenSeesPy command
00783     type used to model it is Hysteresis.
00784     The material model is valid only if the column is continuous.
00785     For more information about the empirical model for the computation of the parameters, see
Skiadopoulos et Al. 2021.
00786     The vectors that forms the matrix used to compute the material model parameters (Kf_Ke_tests,
Cw1_tests, Cfl_tests,
00787         Cw4_tests, Cf4_tests, Cw6_tests, Cf6_tests) are used as global throughout the class to
optimise the program (given the fact that is constant everytime).
00788
00789     @param MaterialModels: Parent abstract class.
00790     """
00791     global Kf_Ke_tests, Cw1_tests, Cfl_tests, Cw4_tests, Cf4_tests, Cw6_tests, Cf6_tests
00792
00793     Kf_Ke_tests = [1.000, 0.153, 0.120, 0.090, 0.059, 0.031, 0.019, 0.009, 0.005, 0.004, 0.000]
00794     Kf_Ke_tests.reverse()
00795     Cw1_tests = [0.96, 0.96, 0.955, 0.94, 0.93, 0.90, 0.89, 0.89, 0.88, 0.88, 0.88]
00796     Cw1_tests.reverse()
00797     Cfl_tests = [0.035, 0.035, 0.033, 0.031, 0.018, 0.015, 0.013, 0.009, 0.009, 0.010, 0.010]
00798     Cfl_tests.reverse()
00799     Cw4_tests = [1.145, 1.145, 1.140, 1.133, 1.120, 1.115, 1.115, 1.11, 1.10, 1.10, 1.10]
00800     Cw4_tests.reverse()
00801     Cf4_tests = [0.145, 0.145, 0.123, 0.111, 0.069, 0.040, 0.040, 0.018, 0.010, 0.012, 0.012]
00802     Cf4_tests.reverse()
00803     Cw6_tests = [1.205, 1.2050, 1.2000, 1.1925, 1.1740, 1.1730, 1.1720, 1.1690, 1.1670, 1.1650,
1.1650]
00804     Cw6_tests.reverse()
00805     Cf6_tests = [0.165, 0.1650, 0.1400, 0.1275, 0.0800, 0.0500, 0.0500, 0.0180, 0.0140, 0.0120,
0.0120]
00806     Cf6_tests.reverse()
00807
00808     def __init__(self, ID: int, d_c, bf_c, tf_c, I_c, d_b, tf_b, Fy, E, t_p,
00809         t_dp = 0.0, a_s = 0.03, pinchx = 0.25, pinchy = 0.75, dmg1 = 0.0, dmg2 = 0.0, beta = 0.0,
safety_factor = False, t_fbp = 0):

```

```

00810         """
00811         Constructor of the class.
00812
00813         @param ID (int): Unique material model ID.
00814         @param d_c (float): Column depth.
00815         @param bf_c (float): Column flange width.
00816         @param tf_c (float): Column flange thickness.
00817         @param I_c (float): Column moment of inertia (strong axis).
00818         @param d_b (float): Beam depth.
00819         @param tf_b (float): Beam flange thickness.
00820         @param Fy (float): Yield strength (if assume continous column, Fy of the web).
00821         @param E (float): Young modulus.
00822         @param t_p (float): Panel zone thickness.
00823         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
00824         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
00825         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
00826         Defaults to 0.25
00827         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
00828         Defaults to 0.75
00829         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
00830         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
00831         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
00832         on ductility, mu-beta. Defaults to 0.0.
00833         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
00834         are used (not test results). Defaults to False.
00835         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to
00836         0.
00837
00838         @exception NegativeValue: ID needs to be a positive integer.
00839         @exception NegativeValue: d_c needs to be positive.
00840         @exception NegativeValue: bf_c needs to be positive.
00841         @exception NegativeValue: tf_c needs to be positive.
00842         @exception NegativeValue: d_b needs to be positive.
00843         @exception NegativeValue: tf_b needs to be positive.
00844         @exception NegativeValue: Fy needs to be positive.
00845         @exception NegativeValue: E needs to be positive.
00846         @exception NegativeValue: t_p needs to be positive.
00847         @exception NegativeValue: a_s needs to be positive.
00848         """
00849         # Check
00850         if ID < 1: raise NegativeValue()
00851         if d_c < 0: raise NegativeValue()
00852         if bf_c < 0: raise NegativeValue()
00853         if tf_c < 0: raise NegativeValue()
00854         if d_b < 0: raise NegativeValue()
00855         if tf_b < 0: raise NegativeValue()
00856         if Fy < 0: raise NegativeValue()
00857         if E < 0: raise NegativeValue()
00858         if t_p < 0: raise NegativeValue()
00859         if a_s < 0: raise NegativeValue()
00860         if t_fbp < 0: raise NegativeValue()
00861
00862         # Arguments
00863         self.IDID = ID
00864         self.d_cd_c = d_c
00865         self.bf_cbf_c = bf_c
00866         self.tf_ctf_c = tf_c
00867         self.I_cI_c = I_c
00868         self.d_bd_b = d_b
00869         self.tf_btf_b = tf_b
00870         self.FyFy = Fy
00871         self.EE = E
00872         self.t_pt_p = t_p
00873         self.t_dpt_dp = t_dp
00874         self.a_sa_s = a_s
00875         self.pinchxpinchx = pinchx
00876         self.pinchypinchy = pinchy
00877         self.dmg1dmg1 = dmg1
00878         self.dmg2dmg2 = dmg2
00879         self.betabeta = beta
00880         if safety_factor:
00881             self.RyRy = 1.2
00882         else:
00883             self.RyRy = 1.0
00884         self.t_fbp_t_fbp = t_fbp
00885
00886         # Initialized the parameters that are dependent from others
00887         self.beam_section_name_tagbeam_section_name_tag = "None"
00888         self.col_section_name_tagcol_section_name_tag = "None"
00889         self.InitializedInitialized = False
00890         self.ReInitReInit()
00891
00892         # Methods
00893         def ReInit(self):
00894             """
00895             Implementation of the homonym abstract method.

```

```

00892         See parent class DataManagement for detailed information.
00893         """
00894         # Check applicability
00895         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
00896
00897         # Memebers
00898         if self.beam_section_name_tagbeam_section_name_tag != "None":
00899             self.beam_section_name_tagbeam_section_name_tag = self.beam_section_name_tagbeam_section_name_tag + "
(modified)"
00900         if self.col_section_name_tagcol_section_name_tag != "None":
00901             self.col_section_name_tagcol_section_name_tag = self.col_section_name_tagcol_section_name_tag + "
(modified)"
00902         self.t_pzt_pz = self.t_pt_p + self.t_dpt_dp
00903         self.GG = self.EE/(2.0 * (1.0 + 0.30)) # Shear Modulus
00904
00905         # Refined computation of the parameters for the backbone curve for the panel zone spring
00906         (Skiadopoulos et al. (2021))
00907         # Panel Zone Elastic Stiffness
00908         self.KsKs = self.t_pzt_pz*(self.d_cd_c-self.tf_ctf_c)*self.GG
00909         self.KbKb =
12.0*self.EE*(self.I_cI_c+self.t_dpt_dp*(self.d_cd_c-2.0*self.tf_ctf_c)**3/12.0)/(self.d_bd_b-0)**2
00910         self.KeKe = self.KsKs*self.KbKb/(self.KsKs+self.KbKb)
00911
00912         # Column Flange Stiffness
00913         self.KsfKsf = 2.0*((self.tf_ctf_c+self.t_fbp_t_fbp)*self.bf_cbf_c+self.GG)
00914         self.KbfKbf =
2.0*(12.0*self.EE*self.bf_cbf_c*(self.tf_ctf_c**3+self.t_fbp_t_fbp**3)/12.0/(self.d_bd_b-0)**2)
00915         self.KfKf = self.KsfKsf*self.KbfKbf/(self.KsfKsf+self.KbfKbf)
00916
00917         # Kf/Ke Calculation for Panel Zone Categorization
00918         self.Kf_KeKf_Ke = self.KfKf/self.KeKe
00919
00920         # Panel Zone Strength Coefficients (results from tests for a_w_eff and a_f_eff)
00921         self.Cw1Cw1 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cw1_tests)
00922         self.Cf1Cf1 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cf1_tests)
00923         self.Cw4Cw4 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cw4_tests)
00924         self.Cf4Cf4 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cf4_tests)
00925         self.Cw6Cw6 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cw6_tests)
00926         self.Cf6Cf6 = np.interp(self.Kf_KeKf_Ke, Kf_Ke_tests, Cf6_tests)
00927
00928         # Panel Zone Model
00929         self.V1V1 =
self.FyFy*self.RyRy/math.sqrt(3)*(self.Cw1Cw1*(self.d_cd_c-self.tf_ctf_c)*self.t_pzt_pz +
self.Cf1Cf1*2*(self.bf_cbf_c-self.t_pt_p)*self.tf_ctf_c)
00930         self.V4V4 =
self.FyFy*self.RyRy/math.sqrt(3)*(self.Cw4Cw4*(self.d_cd_c-self.tf_ctf_c)*self.t_pzt_pz +
self.Cf4Cf4*2*(self.bf_cbf_c-self.t_pt_p)*self.tf_ctf_c)
00931         self.V6V6 =
self.FyFy*self.RyRy/math.sqrt(3)*(self.Cw6Cw6*(self.d_cd_c-self.tf_ctf_c)*self.t_pzt_pz +
self.Cf6Cf6*2*(self.bf_cbf_c-self.t_pt_p)*self.tf_ctf_c)
00932
00933         self.M1M1 = self.V1V1*(self.d_bd_b-self.tf_btf_b)
00934         self.M4M4 = self.V4V4*(self.d_bd_b-self.tf_btf_b)
00935         self.M6M6 = self.V6V6*(self.d_bd_b-self.tf_btf_b)
00936
00937         self.Gamma_1Gamma_1 = self.V1V1/self.KeKe
00938         self.Gamma_4Gamma_4 = 4*self.Gamma_1Gamma_1
00939         self.Gamma_6Gamma_6 = 6*self.Gamma_1Gamma_1
00940
00941         # Data storage for loading/saving
00942         self.UpdateStoredDataUpdateStoredData()
00943
00944         def UpdateStoredData(self):
00945             """
00946             Implementation of the homonym abstract method.
00947             See parent class DataManagement for detailed information.
00948             """
00949             self.datadata = [{"INFO_TYPE", "Skiadopoulos2021"}, # Tag for differentiating different data
00950                             [{"ID", self.IDID},
00951                             [{"beam_section_name_tag", self.beam_section_name_tagbeam_section_name_tag},
00952                             [{"col_section_name_tag", self.col_section_name_tagcol_section_name_tag},
00953                             [{"d_c", self.d_cd_c},
00954                             [{"bf_c", self.bf_cbf_c},
00955                             [{"tf_c", self.tf_ctf_c},
00956                             [{"I_c", self.I_cI_c},
00957                             [{"d_b", self.d_bd_b},
00958                             [{"tf_b", self.tf_btf_b},
00959                             [{"Fy", self.FyFy},
00960                             [{"E", self.EE},
00961                             [{"G", self.GG},
00962                             [{"t_p", self.t_pt_p},
00963                             [{"t_dp", self.t_dpt_dp},
00964                             [{"t_pz", self.t_pzt_pz},
00965                             [{"a_s", self.a_sa_s},
00966                             [{"pinchx", self.pinchxpinchx},
00967                             [{"pinchy", self.pinchypinchy},

```



```

00966         ["dmg1", self.dmg1dmg1],
00967         ["dmg2", self.dmg2dmg2],
00968         ["beta", self.betabeta],
00969         ["Ry", self.RyRy],
00970         ["Ks", self.KsKs],
00971         ["Kb", self.KbKb],
00972         ["Ke", self.KeKe],
00973         ["Ksf", self.KsfKsf],
00974         ["Kbf", self.KbfKbf],
00975         ["Kf", self.KfKf],
00976         ["Kf_Ke", self.Kf_KeKf_Ke],
00977         ["V1", self.V1V1],
00978         ["V4", self.V4V4],
00979         ["V6", self.V6V6],
00980         ["M1", self.M1M1],
00981         ["M4", self.M4M4],
00982         ["M6", self.M6M6],
00983         ["Gamma_1", self.Gamma_1Gamma_1],
00984         ["Gamma_4", self.Gamma_4Gamma_4],
00985         ["Gamma_6", self.Gamma_6Gamma_6],
00986         ["Initialized", self.InitializedInitialized]]
00987
00988
00989     def ShowInfo(self, plot = False, block = False):
00990         """
00991         Implementation of the homonym abstract method.
00992         See parent class DataManagement for detailed information.
00993
00994         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
00995         False.
00996         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
00997         of the program everytime that a plot should pop up). Defaults to False.
00998         """
00999         print("")
01000         print("Requested info for Skiadopoulos 2021 material model Parameters, ID =
01001         {}".format(self.IDID))
01002         print("Sections associated, column: {}".
01003         format(self.col_section_name_tagcol_section_name_tag))
01004         print("Sections associated, beam: {}".
01005         format(self.beam_section_name_tagbeam_section_name_tag))
01006         print("Gamma_1 = {} rad".format(self.Gamma_1Gamma_1))
01007         print("Gamma_4 = {} rad".format(self.Gamma_4Gamma_4))
01008         print("Gamma_6 = {} rad".format(self.Gamma_6Gamma_6))
01009         print("M1 = {} kNm".format(self.M1M1/kNm_unit))
01010         print("M4 = {} kNm".format(self.M4M4/kNm_unit))
01011         print("M6 = {} kNm".format(self.M6M6/kNm_unit))
01012         print("")
01013
01014         if plot:
01015             # Data for plotting
01016             x_axis = np.array([0.0, self.Gamma_1Gamma_1, self.Gamma_4Gamma_4, self.Gamma_6Gamma_6])
01017             y_axis = np.array([0.0, self.M1M1, self.M4M4, self.M6M6])/kNm_unit
01018
01019             fig, ax = plt.subplots()
01020             ax.plot(x_axis, y_axis, 'k-')
01021
01022             ax.set(xlabel='Rotation [rad]', ylabel='Moment [kNm]',
01023                   title='Skiadopoulos 2021 material model (ID={})'.format(self.IDID))
01024             ax.grid()
01025
01026             if block:
01027                 plt.show()
01028
01029     def CheckApplicability(self):
01030         """
01031         Implementation of the homonym abstract method.
01032         See parent class DataManagement for detailed information.
01033         """
01034         Check = True
01035         # No checks
01036         if not Check:
01037             print("The validity of the equations is not fullfilled.")
01038             print("!!!!!! WARNING !!!!!!! Check material model of Skiadopoulos 2021, ID=", self.IDID)
01039             print("")
01040
01041     def Hysteretic(self):
01042         """
01043         Generate the material model Hysteretic (Skiadopoulos 2021) using the computed parameters.
01044         See _Hysteretic function for more information.
01045         """
01046         _Hysteretic(self.IDID, self.M1M1, self.Gamma_1Gamma_1, self.M4M4, self.Gamma_4Gamma_4,
01047                     self.M6M6, self.Gamma_6Gamma_6,
01048                     self.pinchxpinchx, self.pinchypinchy, self.dmg1dmg1, self.dmg2dmg2, self.betabeta)
01049         self.InitializedInitialized = True
01050         self.UpdateStoredDataUpdateStoredData()

```

```

01047
01048
01049 class Skiadopoulos2021SteelISHape(Skiadopoulos2021):
01050     """
01051     Class that is the children of Skiadopoulos2021 and combine the class SteelISHape (section) to
01052     retrieve the information needed.
01053     @param Skiadopoulos2021: Parent class.
01054     """
01055     def __init__(self, ID: int, col: SteelISHape, beam: SteelISHape,
01056         t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False, t_fbp
01057         = 0):
01058         """
01059         Constructor of the class. It passes the arguments into the parent class to generate the
01060         combination of the parent class
01061         and the section class SteelISHape.
01062         The copy of the sections (col and beam) passed are stored in the member variable self.colcol
01063         and self.beambeam.
01064
01065         @param ID (int): Unique material model ID.
01066         @param col (SteelISHape): SteelISHape column section object.
01067         @param beam (SteelISHape): SteelISHape beam section object.
01068         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
01069         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
01070         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
01071         Defaults to 0.25.
01072         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
01073         Defaults to 0.75.
01074         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
01075         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
01076         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
01077         on ductility, mu-beta. Defaults to 0.0.
01078         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
01079         are used (not test results). Defaults to False.
01080         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to
01081         0.
01082         """
01083         self.colcol = deepcopy(col)
01084         self.beambeam = deepcopy(beam)
01085         super().__init__(ID, col.d, col.bf, col.tf, col.Iy, beam.d, beam.tf, col.Fy_web, col.E,
01086             col.tw,
01087             t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta,
01088             safety_factor=safety_factor, t_fbp=t_fbp)
01089         self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
01090         self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
01091         self.UpdateStoredDataUpdateStoredData()
01092
01093 class Skiadopoulos2021RCS(Skiadopoulos2021):
01094     """
01095     WIP: Class that is the children of Skiadopoulos2021 and it's used for the panel zone spring in a
01096     RCS (RC column continous, Steel beam).
01097     @param Skiadopoulos2021: Parent class.
01098     """
01099     def __init__(self, ID: int, beam: SteelISHape, d_col, t_fbp = 0,
01100         t_dp=0, a_s=0.03, pinchx=0.25, pinchy=0.75, dmg1=0, dmg2=0, beta=0, safety_factor=False):
01101         """
01102         Constructor of the class. It passes the arguments into the parent class to generate the
01103         combination of the parent class
01104         and the section class SteelISHape.
01105         The copy of the section (beam) passed is stored in the member variable self.beambeam.
01106
01107         @param ID (int): Unique material model ID.
01108         @param beam (SteelISHape): SteelISHape beam section object.
01109         @param d_col (float): Depth of the RC column (continous)
01110         @param t_fbp (float, optional): Thickness of the face bearing plate (if present). Defaults to
01111         0.
01112         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.0.
01113         @param a_s (float, optional): Strain hardening. Defaults to 0.03.
01114         @param pinchx (float, optional): Pinching factor for strain (or deformation) during reloading.
01115         Defaults to 0.25
01116         @param pinchy (float, optional): Pinching factor for stress (or force) during reloading.
01117         Defaults to 0.75
01118         @param dmg1 (float, optional): Damage due to ductility: D1(mu-1). Defaults to 0.0.
01119         @param dmg2 (float, optional): Damage due to energy: D2(Eii/Eult). Defaults to 0.0.
01120         @param beta (float, optional): Power used to determine the degraded unloading stiffness based
01121         on ductility, mu-beta. Defaults to 0.0.
01122         @param safety_factor (bool, optional): Safety factor used if standard mechanical parameters
01123         are used (not test results). Defaults to False.
01124         """
01125         self.beambeam = deepcopy(beam)
01126         super().__init__(ID, beam.d, beam.bf, beam.tf, beam.Iy, d_col, 0, beam.Fy_web, beam.E,
01127             beam.tw,
01128             t_dp=t_dp, a_s=a_s, pinchx=pinchx, pinchy=pinchy, dmg1=dmg1, dmg2=dmg2, beta=beta,
01129             safety_factor=safety_factor, t_fbp=t_fbp)
01130         self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag

```

```

01114         self.UpdateStoredDataUpdateStoredData()
01115
01116
01117 class UnconfMander1988(MaterialModels):
01118     """
01119     Class that stores functions and material properties of a RC rectangular or circular section
01120     with Mander 1988 as the material model for the unconfined reinforced concrete and the
01121     OpenSeesPy command type used to model it is Concrete04 or Concrete01.
01122     For more information about the empirical model for the computation of the parameters, see Mander
01123     et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.
01124
01125     @param MaterialModels: Parent abstract class.
01126     """
01127     def __init__(self, ID: int, fc, Ec, ec = 1, ecp = 1, fct = -1, et = -1, beta = 0.1):
01128         """
01129         Constructor of the class.
01130
01131         @param ID (int): Unique material model ID.
01132         @param fc (float): Compressive concrete yield strength (needs to be negative).
01133         @param Ec (float): Young modulus.
01134         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01135         according to Karthik and Mander 2011.
01136         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01137         to Mander 1988.
01138         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01139         according to SIA 262:2012.
01140         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01141         according to SIA 262:2012.
01142         @param beta (float, optional): Loading point value defining the exponential curve parameter to
01143         define the residual stress.
01144         Defaults to 0.1 (according to OpenSeesPy documentation)
01145
01146         @exception NegativeValue: ID needs to be a positive integer.
01147         @exception PositiveValue: fc needs to be negative.
01148         @exception NegativeValue: Ec needs to be positive.
01149         @exception PositiveValue: ec needs to be negative if different from 1.
01150         @exception PositiveValue: ecp needs to be positive if different from 1.
01151         @exception NegativeValue: fct needs to be positive if different from -1.
01152         @exception NegativeValue: et needs to be positive if different from -1.
01153         """
01154         # Check
01155         if ID < 0: raise NegativeValue()
01156         if fc > 0: raise PositiveValue()
01157         if Ec < 0: raise NegativeValue()
01158         if ec != 1 and ec > 0: raise PositiveValue()
01159         if ecp != 1 and ecp > 0: raise PositiveValue()
01160         if fct != -1 and fct < 0: raise NegativeValue()
01161         if et != -1 and et < 0: raise NegativeValue()
01162
01163         # Arguments
01164         self.IDID = ID
01165         self.fcfc = fc
01166         self.EcEc = Ec
01167         self.betabeta = beta
01168
01169         # Initialized the parameters that are dependent from others
01170         self.section_name_tagsection_name_tag = "None"
01171         self.InitializedInitialized = False
01172         self.ReInitReInit(ec, ecp, fct, et)
01173
01174     # Methods
01175     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
01176         """
01177         Implementation of the homonym abstract method.
01178         See parent class DataManagement for detailed information.
01179
01180         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01181         according to Karthik and Mander 2011.
01182         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01183         to Mander 1988.
01184         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01185         according to SIA 262:2012.
01186         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01187         according to SIA 262:2012.
01188         """
01189         # Check applicability
01190         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
01191
01192         # Arguments
01193         self.ec = self.Compute_ecCompute_ec() if ec == 1 else ec
01194         self.ecp = self.Compute_ecpCompute_ecp() if ecp == 1 else ecp
01195         self.fct = self.Compute_fctCompute_fct() if fct == -1 else fct
01196         self.et = self.Compute_etCompute_et() if et == -1 else et
01197
01198         # Members
01199         self.ecu = self.Compute_ecuCompute_ecu()
01200         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =

```

```

self.section_name_tagsection_name_tag + " (modified)"

01190
01191     # Data storage for loading/saving
01192     self.UpdateStoredDataUpdateStoredData()
01193
01194
01195     def UpdateStoredData(self):
01196         """
01197         Implementation of the homonym abstract method.
01198         See parent class DataManagement for detailed information.
01199         """
01200         self.datadata = [{"INFO_TYPE", "UnconfMander1988"}, # Tag for differentiating different data
01201                           [{"ID", self.IDID},
01202                            ["section_name_tag", self.section_name_tagsection_name_tag],
01203                            ["fc", self.fcfc],
01204                            ["Ec", self.EcEc],
01205                            ["ec", self.ecec],
01206                            ["ecp", self.ecpecp],
01207                            ["ecu", self.ecuecu],
01208                            ["fct", self.fctfct],
01209                            ["et", self.etet],
01210                            ["beta", self.betabeta],
01211                            ["Initialized", self.InitializedInitialized]]
01212
01213
01214     def ShowInfo(self, plot = False, block = False, concrete04 = True):
01215         """
01216         Implementation of the homonym abstract method.
01217         See parent class DataManagement for detailed information.
01218
01219         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
01220         False.
01221         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
01222         of the program everytime that a plot should pop up). Defaults to False.
01223         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if
01224         False. Defaults to True.
01225         """
01226         print("")
01227         print("Requested info for Unconfined Mander 1988 material model Parameters, ID =
01228         {}".format(self.IDID))
01229         print("Section associated: {}".format(self.section_name_tagsection_name_tag))
01230         print('Concrete strength fc = {} MPa'.format(self.fcfc/MPa_unit))
01231         print('Strain at maximal strength ec = {}'.format(self.ecec))
01232         print('Maximal strain ecu = {}'.format(self.ecuecu))
01233         print("")
01234
01235         if plot:
01236             fig, ax = plt.subplots()
01237             if concrete04:
01238                 PlotConcrete04(self.fcfc, self.EcEc, self.ecec, self.ecuecu, "U", ax, self.IDID)
01239             else:
01240                 PlotConcrete01(self.fcfc, self.ecec, 0, self.ecuecu, ax, self.IDID)
01241
01242             if block:
01243                 plt.show()
01244
01245     def CheckApplicability(self):
01246         """
01247         Implementation of the homonym abstract method.
01248         See parent class MaterialModels for detailed information.
01249         """
01250         Check = True
01251         if self.fcfc < -110*MPa_unit: # Deierlein 1999
01252             Check = False
01253         print("With High Strength concrete (< -110 MPa), a better material model should be used
01254         (see Abdesselam et Al. 2019)")
01255         if not Check:
01256             print("The validity of the equations is not fullfilled.")
01257             print("!!!!!! WARNING !!!!!!! Check material model of Unconfined Mander 1988, ID=",
01258             self.IDID)
01259             print("")
01260
01261     def Compute_ec(self):
01262         """
01263         Method that computes the compressive concrete yield strain.
01264         For more information, see Karthik and Mander 2011.
01265
01266         @returns float: Strain
01267         """
01268         # return -0.002 # Alternative: Mander et Al. 1988
01269         return -0.0015 + self.fcfc/MPa_unit/70000 # Karthik Mander 2011
01270
01271     def Compute_ecp(self):
01272         """
01273         Method that computes the compressive concrete spalling strain.

```

```

01270         For more information, see Mander et Al. 1988.
01271
01272         @returns float: Strain
01273         """
01274         return 2.0*self.ecec
01275
01276
01277     def Compute_fct(self):
01278         """
01279         Method that computes the tensile concrete yield stress.
01280         For more information, see SIA 262:2012.
01281
01282         @returns float: Stress.
01283         """
01284         return 0.30 * math.pow(-self.fcfc/MPa_unit, 2/3) * MPa_unit
01285
01286
01287     def Compute_et(self):
01288         """
01289         Method that computes the tensile concrete yield strain.
01290         For more information, see Mander et Al. 1988 (eq 45).
01291
01292         @returns float: Strain.
01293         """
01294         return self.fctfct/self.EcEc
01295
01296
01297     def Compute_ecu(self):
01298         """
01299         Method that computes the compressive concrete failure strain.
01300         For more information, see Karthik and Mander 2011.
01301
01302         @returns float: Strain
01303         """
01304         # return -0.004 # Alternative: Mander et Al. 1988
01305         return -0.012 - 0.0001 * self.fcfc/MPa_unit # Karthik Mander 2011
01306
01307     def Concrete01(self):
01308         """
01309         Generate the material model Concrete01 for unconfined concrete using the computed parameters.
01310         See _Concrete01 function for more information. Use this method or Concrete04, not both (only
one material model for ID).
01311         """
01312         _Concrete01(self.IDID, self.ecec, self.fcfc, self.ecuecu)
01313         self.InitializedInitialized = True
01314         self.UpdateStoredDataUpdateStoredData()
01315
01316
01317     def Concrete04(self):
01318         """
01319         Generate the material model Concrete04 for unconfined concrete (Mander 1988) using the
computed parameters.
01320         See _Concrete04 function for more information. Use this method or Concrete01, not both (only
one material model for ID).
01321         """
01322         _Concrete04(self.IDID, self.fcfc, self.ecec, self.ecuecu, self.EcEc, self.fctfct, self.etet,
self.betabeta)
01323         self.InitializedInitialized = True
01324         self.UpdateStoredDataUpdateStoredData()
01325
01326
01327 class UnconfMander1988RCRectShape(UnconfMander1988):
01328     """
01329     Class that is the children of UnconfMander1988 and combine the class RCRectShape (section) to
retrieve the information needed.
01330
01331     @param UnconfMander1988: Parent class.
01332     """
01333     def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
01334         """
01335         Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
and the section class RCRectShape.
01336         The copy of the section passed is stored in the member variable self.sectionsection.
01337
01338         @param ID (int): Unique material model ID.
01339         @param section (RCRectShape): RCRectShape section object.
01340         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
01341         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
01342         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01343         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01344         @param beta (float, optional): Loading point value defining the exponential curve parameter to
define the residual stress.

```

```

01346         Defaults to 0.1 (according to OpenSeesPy documentation)
01347         """
01348         self.sectionsection = deepcopy(section)
01349         super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
01350         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
01351         self.UpdateStoredDataUpdateStoredData()
01352
01353
01354 class UnconfMander1988RCCircShape(UnconfMander1988):
01355     """
01356     Class that is the children of UnconfMander1988 and combine the class RCCircShape (section) to
01357     retrieve the information needed.
01358
01359     @param UnconfMander1988: Parent class.
01360     """
01361     def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, beta=0.1):
01362         """
01363         Constructor of the class. It passes the arguments into the parent class to generate the
01364         combination of the parent class
01365         and the section class RCCircShape.
01366         The copy of the section passed is stored in the member variable self.sectionsection.
01367
01368         @param ID (int): Unique material model ID.
01369         @param section (RCCircShape): RCCircShape section object.
01370         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01371         according to Karthik and Mander 2011.
01372         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01373         to Mander 1988.
01374         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01375         according to SIA 262:2012.
01376         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01377         according to SIA 262:2012.
01378         @param beta (float, optional): Loading point value defining the exponential curve parameter to
01379         define the residual stress.
01380         Defaults to 0.1 (according to OpenSeesPy documentation)
01381         """
01382         self.sectionsection = deepcopy(section)
01383         super().__init__(ID, section.fc, section.Ec, ec=ec, ecp=ecp, fct=fct, et=et, beta=beta)
01384         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
01385         self.UpdateStoredDataUpdateStoredData()
01386
01387
01388 class ConfMander1988Rect(MaterialModels):
01389     """
01390     Class that stores functions and material properties of a RC rectangular section
01391     with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy
01392     command type used to model it is Concrete04 or Concrete01.
01393     For more information about the empirical model for the computation of the parameters, see Mander
01394     et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.
01395     The array array_fl2 and curve_curve_fl1 are the parameter of the digitized table used to
01396     extrapolate the confinement factor;
01397     they are used as global throughout the ConfMander1988Rect material model to optimise the
01398     program (given the fact that is constant everytime).
01399
01400     @param MaterialModels: Parent abstract class.
01401     """
01402     global array_fl2, curve_fl1
01403
01404     curve_fl1 = np.arange(0, 0.3+0.02, 0.02)
01405     array_fl2 = [None] * len(curve_fl1)
01406
01407     array_fl2[0] = [[1.0, 0],
01408                    [1.0026455026455026, 6.5359477124183E-4],
01409                    [1.0423280423280423, 0.01699346405228758],
01410                    [1.0846560846560847, 0.037254901960784306],
01411                    [1.119047619047619, 0.05686274509803921],
01412                    [1.1455026455026456, 0.0784313725490196],
01413                    [1.1666666666666667, 0.09607843137254903],
01414                    [1.193121693121693, 0.11830065359477124],
01415                    [1.208994708994709, 0.1392156862745098],
01416                    [1.2248677248677249, 0.15751633986928104],
01417                    [1.2380952380952381, 0.17712418300653593],
01418                    [1.2513227513227514, 0.19673202614379084],
01419                    [1.2645502645502646, 0.21699346405228756],
01420                    [1.2724867724867726, 0.23660130718954248],
01421                    [1.2804232804232805, 0.2594771241830065],
01422                    [1.2883597883597884, 0.2777777777777778],
01423                    [1.2936507936507937, 0.3]]
01424
01425     array_fl2[1] = [[1.1349206349206349, 0.01895424836601307],
01426                    [1.1825396825396826, 0.03790849673202614],
01427                    [1.2222222222222223, 0.05686274509803921],
01428                    [1.2513227513227514, 0.07777777777777778],
01429                    [1.2724867724867726, 0.09738562091503267],
01430                    [1.291005291005291, 0.11895424836601308],
01431                    [1.3174603174603174, 0.13856209150326795],
01432                    [1.335978835978836, 0.1588235294117647]]

```

```
01422         [1.3518518518518519, 0.17777777777777776],
01423         [1.3677248677248677, 0.19738562091503267],
01424         [1.3783068783068784, 0.2176470588235294],
01425         [1.3941798941798942, 0.238562091503268],
01426         [1.41005291005291, 0.2594771241830065],
01427         [1.4126984126984126, 0.28104575163398693],
01428         [1.4232804232804233, 0.3]]
01429
01430     array_fl2[2] = [[1.246031746031746, 0.037254901960784306],
01431                    [1.298941798941799, 0.05751633986928104],
01432                    [1.335978835978836, 0.07712418300653595],
01433                    [1.3650793650793651, 0.09869281045751634],
01434                    [1.3888888888888888, 0.11699346405228757],
01435                    [1.4153439153439153, 0.1392156862745098],
01436                    [1.439153439153439, 0.1568627450980392],
01437                    [1.4603174603174602, 0.17712418300653593],
01438                    [1.4735449735449735, 0.1980392156862745],
01439                    [1.4894179894179893, 0.21633986928104573],
01440                    [1.5052910052910053, 0.23790849673202616],
01441                    [1.5211640211640212, 0.2581699346405229],
01442                    [1.5317460317460316, 0.2777777777777778],
01443                    [1.5423280423280423, 0.3]]
01444
01445     array_fl2[3] = [[1.3544973544973544, 0.05686274509803921],
01446                    [1.3994708994708995, 0.07777777777777777],
01447                    [1.4417989417989419, 0.09738562091503267],
01448                    [1.4735449735449735, 0.11699346405228757],
01449                    [1.4947089947089947, 0.13790849673202613],
01450                    [1.5238095238095237, 0.15751633986928104],
01451                    [1.5423280423280423, 0.17777777777777776],
01452                    [1.560846560846561, 0.1980392156862745],
01453                    [1.5820105820105819, 0.21633986928104573],
01454                    [1.5952380952380953, 0.2372549019607843],
01455                    [1.6137566137566137, 0.2588235294117647],
01456                    [1.626984126984127, 0.27908496732026145],
01457                    [1.6375661375661377, 0.3]]
01458
01459     array_fl2[4] = [[1.455026455026455, 0.07647058823529412],
01460                    [1.5, 0.09738562091503267],
01461                    [1.5423280423280423, 0.1196078431372549],
01462                    [1.574074074074074, 0.13856209150326795],
01463                    [1.6058201058201058, 0.1588235294117647],
01464                    [1.6296296296296298, 0.17777777777777776],
01465                    [1.6534391534391535, 0.1980392156862745],
01466                    [1.671957671957672, 0.21699346405228756],
01467                    [1.6904761904761905, 0.238562091503268],
01468                    [1.7063492063492065, 0.2588235294117647],
01469                    [1.716931216931217, 0.27908496732026145],
01470                    [1.7248677248677249, 0.3]]
01471
01472     array_fl2[5] = [[1.5634920634920635, 0.09869281045751634],
01473                    [1.6058201058201058, 0.11895424836601308],
01474                    [1.6428571428571428, 0.13790849673202613],
01475                    [1.6746031746031746, 0.1588235294117647],
01476                    [1.701058201058201, 0.17908496732026144],
01477                    [1.7222222222222223, 0.19673202614379084],
01478                    [1.7433862433862433, 0.2176470588235294],
01479                    [1.7698412698412698, 0.2392156862745098],
01480                    [1.783068783068783, 0.2581699346405229],
01481                    [1.798941798941799, 0.27908496732026145],
01482                    [1.8095238095238095, 0.3]]
01483
01484     array_fl2[6] = [[1.6507936507936507, 0.11633986928104575],
01485                    [1.693121693121693, 0.13856209150326795],
01486                    [1.7328042328042328, 0.15751633986928104],
01487                    [1.7645502645502646, 0.17843137254901958],
01488                    [1.7910052910052912, 0.1980392156862745],
01489                    [1.8148148148148149, 0.2176470588235294],
01490                    [1.8333333333333335, 0.23790849673202616],
01491                    [1.8571428571428572, 0.2581699346405229],
01492                    [1.8677248677248677, 0.2797385620915033],
01493                    [1.8915343915343916, 0.3]]
01494
01495     array_fl2[7] = [[1.753968253968254, 0.14052287581699346],
01496                    [1.7883597883597884, 0.15816993464052287],
01497                    [1.8174603174603174, 0.17843137254901958],
01498                    [1.8412698412698414, 0.19738562091503267],
01499                    [1.8677248677248677, 0.21699346405228756],
01500                    [1.8835978835978837, 0.2372549019607843],
01501                    [1.9047619047619047, 0.257516339869281],
01502                    [1.925925925925926, 0.27908496732026145],
01503                    [1.9417989417989419, 0.3]]
01504
01505     array_fl2[8] = [[1.8386243386243386, 0.16013071895424835],
01506                    [1.8703703703703702, 0.17908496732026144],
01507                    [1.8994708994708995, 0.1980392156862745],
01508                    [1.925925925925926, 0.2176470588235294],
```

```

01509         [1.9497354497354498, 0.23790849673202616],
01510         [1.9761904761904763, 0.2588235294117647],
01511         [1.992063492063492, 0.27908496732026145],
01512         [2.0132275132275135, 0.3]]
01513
01514     array_fl2[9] = [[1.9179894179894181, 0.1823529411764706],
01515                    [1.939153439153439, 0.19869281045751636],
01516                    [1.9682539682539684, 0.21895424836601307],
01517                    [1.992063492063492, 0.238562091503268],
01518                    [2.0132275132275135, 0.2568627450980392],
01519                    [2.0396825396825395, 0.27908496732026145],
01520                    [2.060846560846561, 0.3]]
01521
01522     array_fl2[10] = [[1.9761904761904763, 0.19673202614379084],
01523                    [2.007936507936508, 0.21633986928104573],
01524                    [2.0343915343915344, 0.2372549019607843],
01525                    [2.066137566137566, 0.257516339869281],
01526                    [2.08994708994709, 0.2784313725490196],
01527                    [2.111111111111111, 0.3]]
01528
01529     array_fl2[11] = [[2.044973544973545, 0.21633986928104573],
01530                    [2.0767195767195767, 0.238562091503268],
01531                    [2.1084656084656084, 0.2581699346405229],
01532                    [2.134920634920635, 0.28039215686274505],
01533                    [2.158730158730159, 0.3]]
01534
01535     array_fl2[12] = [[2.113756613756614, 0.2372549019607843],
01536                    [2.1455026455026456, 0.257516339869281],
01537                    [2.1719576719576716, 0.2797385620915033],
01538                    [2.193121693121693, 0.3]]
01539
01540     array_fl2[13] = [[2.177248677248677, 0.2581699346405229],
01541                    [2.2063492063492065, 0.27908496732026145],
01542                    [2.2275132275132274, 0.3]]
01543
01544     array_fl2[14] = [[2.2407407407407405, 0.2784313725490196],
01545                    [2.261904761904762, 0.3]]
01546
01547     array_fl2[15] = [[2.2962962962962963, 0.3]]
01548
01549     def __init__(self, ID: int, bc, dc, Ac, fc, Ec, nr_bars, D_bars, wx_top: np.ndarray, wx_bottom:
np.ndarray, wy: np.ndarray, s, D_hoops, rho_s_x, rho_s_y, fs,
01550                  ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
01551         """
01552         Constructor of the class.
01553
01554         @param ID (int): Unique material model ID.
01555         @param bc (float): Width of the confined core (from the centerline of the hoops, according to
Mander et Al. 1988).
01556         @param dc (float): Depth of the confined core (from the centerline of the hoops, according to
Mander et Al. 1988).
01557         @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
01558         @param fc (float): Compressive concrete yield strength (needs to be negative).
01559         @param Ec (float): Young modulus.
01560         @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent
nr_bars with different reinforcement areas).
01561         @param D_bars (float): Diameter of the vertical reinforcing bars.
01562         @param wx_top (np.ndarray): Vector of 1 dimension that defines the distance between top
vertical bars in x direction (NOT CENTERLINE DISTANCES).
01563         @param wx_bottom (np.ndarray): Vector of 1 dimension that defines the distance between bottom
vertical bars in x direction (NOT CENTERLINE DISTANCES).
01564         @param wy (np.ndarray): Vector of 1 dimension that defines the distance between vertical bars
in y direction (lateral) (NOT CENTERLINE DISTANCES).
01565         @param s (float): Vertical spacing between hoops.
01566         @param D_hoops (float): Diameter of hoops.
01567         @param rho_s_x (float): Ratio of the transversal area of the hoops to the associated concrete
area in the x direction.
01568         @param rho_s_y (float): Ratio of the transversal area of the hoops to the associated concrete
area in the y direction.
01569         @param fs (float): Yield stress for the hoops.
01570         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
01571         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
01572         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01573         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01574         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
according to Mander 1988.
01575         @param beta (float, optional): Loading point value defining the exponential curve parameter to
define the residual stress.
01576             Defaults to 0.1 (according to OpenSeesPy documentation)
01577
01578         @exception NegativeValue: ID needs to be a positive integer.
01579         @exception NegativeValue: bc needs to be positive.
01580         @exception NegativeValue: dc needs to be positive.

```



```

01581         @exception NegativeValue: Ac needs to be positive.
01582         @exception PositiveValue: fc needs to be negative.
01583         @exception NegativeValue: Ec needs to be positive.
01584         @exception NegativeValue: nr_bars needs to be positive.
01585         @exception NegativeValue: D_bars needs to be positive.
01586         @exception NegativeValue: s needs to be positive.
01587         @exception NegativeValue: D_hoops needs to be positive.
01588         @exception NegativeValue: rho_s_x needs to be positive.
01589         @exception NegativeValue: rho_s_y needs to be positive.
01590         @exception NegativeValue: fs needs to be positive.
01591         @exception PositiveValue: ec needs to be negative if different from 1.
01592         @exception PositiveValue: ecp needs to be negative if different from 1.
01593         @exception NegativeValue: fct needs to be positive if different from -1.
01594         @exception NegativeValue: et needs to be positive if different from -1.
01595         @exception NegativeValue: esu needs to be positive if different from -1.
01596         """
01597         # Check
01598         if ID < 1: raise NegativeValue()
01599         if bc < 0: raise NegativeValue()
01600         if dc < 0: raise NegativeValue()
01601         if Ac < 0: raise NegativeValue()
01602         if fc > 0: raise PositiveValue()
01603         if Ec < 0: raise NegativeValue()
01604         if nr_bars < 0: raise NegativeValue()
01605         if D_bars < 0: raise NegativeValue()
01606         if s < 0: raise NegativeValue()
01607         if D_hoops < 0: raise NegativeValue()
01608         if rho_s_x < 0: raise NegativeValue()
01609         if rho_s_y < 0: raise NegativeValue()
01610         if fs < 0: raise NegativeValue()
01611         if ec != 1 and ec > 0: raise PositiveValue()
01612         if ecp != 1 and ecp > 0: raise PositiveValue()
01613         if fct != -1 and fct < 0: raise NegativeValue()
01614         if et != -1 and et < 0: raise NegativeValue()
01615         if esu != -1 and esu < 0: raise NegativeValue()
01616
01617         # Arguments
01618         self.IDID = ID
01619         self.bcbc = bc
01620         self.dcdc = dc
01621         self.AcAc = Ac
01622         self.fcfc = fc
01623         self.EcEc = Ec
01624         self.nr_barsnr_bars = nr_bars
01625         self.D_barsD_bars = D_bars
01626         self.wx_topwx_top = copy(wx_top)
01627         self.wx_bottomwx_bottom = copy(wx_bottom)
01628         self.wywy = copy(wy)
01629         self.ss = s
01630         self.D_hoopsD_hoops = D_hoops
01631         self.rho_s_xrho_s_x = rho_s_x
01632         self.rho_s_yrho_s_y = rho_s_y
01633         self.fsfs = fs
01634         self.esuesu = 0.05 if esu == -1 else esu # Mander 1988
01635         self.betabeta = beta
01636
01637         # Initialized the parameters that are dependent from others
01638         self.section_name_tagsection_name_tag = "None"
01639         self.InitializedInitialized = False
01640         self.ReInitReInit(ec, ecp, fct, et)
01641
01642     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
01643         """
01644         Implementation of the homonym abstract method.
01645         See parent class DataManagement for detailed information.
01646
01647         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
01648         according to Karthik and Mander 2011.
01649         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
01650         to Mander 1988.
01651         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01652         according to SIA 262:2012.
01653         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
01654         according to SIA 262:2012.
01655         """
01656         # Check applicability
01657         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
01658
01659         # Arguments
01660         self.ec = self.Compute_ecCompute_ec() if ec == 1 else ec
01661         self.ecp = self.Compute_ecpCompute_ecp() if ecp == 1 else ecp
01662         self.fct = self.Compute_fctCompute_fct() if fct == -1 else fct
01663         self.et = self.Compute_etCompute_et() if et == -1 else et
01664
01665         # Members (according to Mander 1988, confined concrete)
01666         self.ecu = self.Compute_ecuCompute_ecu()
01667         self.AiAi = self.ComputeAiComputeAi()

```

```

01664         self.AeAe = (self.AcAc - self.AiAi) * (1.0 - (self.ss-self.D_hoopsD_hoops)/2.0/self.bcbc)*(1.0
- (self.ss-self.D_hoopsD_hoops)/2.0/self.dcdc)
01665         self.rho_ccrho_cc = self.nr_barsnr_bars*self.D_barsD_bars**2/4.0*math.pi / self.AcAc
01666         self.AccAcc = self.AcAc*(1.0-self.rho_ccrho_cc)
01667         self.keke = self.AeAe/self.AccAcc
01668         self.fl_xfl_x = -self.rho_s_xrho_s_x * self.fsfs
01669         self.fl_yfl_y = -self.rho_s_yrho_s_y * self.fsfs
01670         self.K_comboK_combo = self.ComputeConfinementFactorComputeConfinementFactor()
01671         self.fccfcc = self.fcfc * self.K_comboK_combo
01672         self.ecccecc = self.Compute_eccCompute_ecc()
01673         self.eccueccu = self.Compute_eccuCompute_eccu()
01674         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"

01675
01676         # Data storage for loading/saving
01677         self.UpdateStoredDataUpdateStoredData()
01678
01679
01680     # Methods
01681     def UpdateStoredData(self):
01682         """
01683         Implementation of the homonym abstract method.
01684         See parent class DataManagement for detailed information.
01685         """
01686         self.datadata = [{"INFO_TYPE", "ConfMander1988rect"}, # Tag for differentiating different data
01687             ["ID", self.IDID],
01688             ["section_name_tag", self.section_name_tagsection_name_tag],
01689             ["bc", self.bcbc],
01690             ["dc", self.dcdc],
01691             ["Ac", self.AcAc],
01692             ["fc", self.fcfc],
01693             ["Ec", self.EcEc],
01694             ["ec", self.ecec],
01695             ["ecp", self.ecpecp],
01696             ["ecu", self.ecuecu],
01697             ["fct", self.fctfct],
01698             ["et", self.etet],
01699             ["fcc", self.fccfcc],
01700             ["ecc", self.ecccecc],
01701             ["eccu", self.eccueccu],
01702             ["beta", self.betabeta],
01703             ["nr_bars", self.nr_barsnr_bars],
01704             ["D_bars", self.D_barsD_bars],
01705             ["wx_top", self.wx_topwx_top],
01706             ["wx_bottom", self.wx_bottomwx_bottom],
01707             ["wy", self.wywy],
01708             ["s", self.ss],
01709             ["D_hoops", self.D_hoopsD_hoops],
01710             ["rho_s_x", self.rho_s_xrho_s_x],
01711             ["rho_s_y", self.rho_s_yrho_s_y],
01712             ["fs", self.fsfs],
01713             ["esu", self.esuesu],
01714             ["Ai", self.AiAi],
01715             ["Ae", self.AeAe],
01716             ["rho_cc", self.rho_ccrho_cc],
01717             ["Acc", self.AccAcc],
01718             ["ke", self.keke],
01719             ["fl_x", self.fl_xfl_x],
01720             ["fl_y", self.fl_yfl_y],
01721             ["K_combo", self.K_comboK_combo],
01722             ["Initialized", self.InitializedInitialized]]
01723
01724
01725     def ShowInfo(self, plot = False, block = False, concrete04 = True):
01726         """
01727         Implementation of the homonym abstract method.
01728         See parent class DataManagement for detailed information.
01729
01730         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
01731         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
01732         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if
False. Defaults to True.
01733         """
01734         print("")
01735         print("Requested info for Confined Mander 1988 (rectangular) material model Parameters, ID =
{}".format(self.IDID))
01736         print("Section associated: {}".format(self.section_name_tagsection_name_tag))
01737         print('Concrete strength fc = {} MPa'.format(self.fcfc/MPa_unit))
01738         print('Concrete strength confined fcc = {} MPa'.format(self.fccfcc/MPa_unit))
01739         print('Strain at maximal strength ec = {}'.format(self.ecec))
01740         print('Strain at maximal strength confined ecc = {}'.format(self.ecccecc))
01741         print('Maximal strain ecu = {}'.format(self.ecuecu))
01742         print('Maximal strain confined eccu = {}'.format(self.eccueccu))
01743         print("")
01744

```

```

01745         if plot:
01746             fig, ax = plt.subplots()
01747             if concrete04:
01748                 PlotConcrete04(self.fccfcc, self.EcEc, self.eccecc, self.eccueccu, "C", ax, self.IDID)
01749             else:
01750                 PlotConcrete01(self.fccfcc, self.eccecc, 0.0, self.eccueccu, ax, self.IDID)
01751
01752         if block:
01753             plt.show()
01754
01755
01756     def CheckApplicability(self):
01757         """
01758         Implementation of the homonym abstract method.
01759         See parent class MaterialModels for detailed information.
01760         """
01761         Check = True
01762         if self.fccfc < -110*MPa_unit: # Deierlein 1999
01763             Check = False
01764             print("With High Strength concrete (< -110 MPa), a better material model should be used
01765 (see Abdesselam et Al. 2019)")
01766         if not Check:
01767             print("The validity of the equations is not fullfilled.")
01768             print("!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=",
01769 self.IDID)
01770             print("")
01771
01772     def Compute_ec(self):
01773         """
01774         Method that computes the compressive concrete yield strain.
01775         For more information, see Karthik and Mander 2011.
01776
01777         @returns float: Strain
01778         """
01779         # return -0.002 # Alternative: Mander et Al. 1988
01780         return -0.0015 + self.fccfc/MPa_unit/70000 # Karthik Mander 2011
01781
01782     def Compute_ecp(self):
01783         """
01784         Method that computes the compressive concrete spalling strain.
01785         For more information, see Mander et Al. 1988.
01786
01787         @returns float: Strain
01788         """
01789         return 2.0*self.ececc
01790
01791     def Compute_fct(self):
01792         """
01793         Method that computes the tensile concrete yield stress.
01794         For more information, see SIA 262:2012. Assume that the confinement do not play an essential
01795 role in tension.
01796
01797         @returns float: Stress.
01798         """
01799         return 0.30 * math.pow(-self.fccfc/MPa_unit, 2/3) * MPa_unit
01800
01801     def Compute_et(self):
01802         """
01803         Method that computes the tensile concrete yield strain.
01804         For more information, see Mander et Al. 1988 (eq 45).
01805
01806         @returns float: Strain.
01807         """
01808         return self.fctfct/self.EcEc
01809
01810     def Compute_ecu(self):
01811         """
01812         Method that computes the compressive concrete failure strain.
01813         For more information, see Karthik and Mander 2011.
01814
01815         @returns float: Strain
01816         """
01817         # return -0.004 # Alternative: Mander et Al. 1988
01818         return -0.012 - 0.0001 * self.fccfc/MPa_unit # Karthik Mander 2011
01819
01820     def Compute_ecc(self):
01821         """
01822         Method that computes the compressive confined concrete yield strain.
01823         For more information, see Karthik and Mander 2011.
01824
01825         @returns float: Strain
01826
01827
01828

```

```

01829         """
01830         return (1.0 + 5.0 * (self.K_comboK_combo-1.0)) * self.ecec # Karthik Mander 2011
01831
01832
01833     def Compute_eccu(self):
01834         """
01835         Method that computes the compressive confined concrete failure strain.
01836         For more information, see Karthik and Mander 2011.
01837
01838         @returns float: Strain
01839         """
01840         # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative:
Prof. Katrin Beyer
01841         return 5*self.eccecc # Karthik Mander 2011
01842
01843
01844     def ComputeAi(self):
01845         """
01846         Method that computes the ineffectual area.
01847         For more information, see Mander et Al. 1988.
01848
01849         @returns float: Area.
01850         """
01851         return ( np.sum(np.multiply(self.wywy, self.wywy))*2.0 +
01852                 np.sum(np.multiply(self.wx_topwx_top, self.wx_topwx_top)) +
01853                 np.sum(np.multiply(self.wx_bottomwx_bottom, self.wx_bottomwx_bottom)) ) / 6.0
01854
01855
01856     def ComputeConfinementFactor(self):
01857         """
01858         Method that computes the confinement factor using the digitized table from Mander et Al. 1988
that
01859         extrapolates the factor using the lateral confining stress in the two direction.
01860
01861         @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
01862         @exception NoApplicability: The table from Mander accept ratio of fl/fc smaller than 0.3.
01863         @exception NegativeValue: fl1_ratio needs to be positive.
01864         @exception NegativeValue: fl2_ratio needs to be positive.
01865
01866         @returns float: Confinement factor.
01867         """
01868         if self.fl_xfl_x == self.fl_yfl_y:
01869             return -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_xfl_x*self.keke/self.fcfc) -
2.0*self.fl_xfl_x*self.keke/self.fcfc # in Mander, it has a prime
01870         else:
01871             fl2_ratio = max(self.fl_xfl_x*self.keke/self.fcfc, self.fl_yfl_y*self.keke/self.fcfc)
01872             fl1_ratio = min(self.fl_xfl_x*self.keke/self.fcfc, self.fl_yfl_y*self.keke/self.fcfc)
01873
01874             if fl1_ratio > 0.3: raise NoApplicability()
01875             if fl2_ratio > 0.3: raise NoApplicability()
01876             if fl1_ratio < 0: raise NegativeValue()
01877             if fl2_ratio < 0: raise NegativeValue()
01878
01879             # choose one or two curves
01880             for ii, fl1 in enumerate(curve_fl1):
01881                 if fl1 == fl1_ratio:
01882                     # one curve
01883                     # choose curve
01884                     curve_fl2 = [curve for ii, curve in enumerate(array_fl2) if index[ii]][0]
01885                     curve_fl2 = array_fl2[ii]
01886
01887                     # Take value (interpolate)
01888                     K = [item[0] for item in curve_fl2]
01889                     fl2 = [item[1] for item in curve_fl2]
01890                     K_res = np.interp(fl2_ratio, fl2, K)
01891
01892                     #TODO: to check fuction:
01893                     # fig, ax = plt.subplots()
01894                     # ax.plot(fl2, K, 'k-')
01895                     # ax.scatter(fl2_ratio, K_res, color='k')
01896                     # ax.grid()
01897                     # plt.show()
01898                     return K_res
01899
01900             # two curves
01901             if fl1 > fl1_ratio:
01902                 fl1_max = fl1
01903                 fl1_min = curve_fl1[ii-1]
01904                 curve_fl2_max = array_fl2[ii]
01905                 curve_fl2_min = array_fl2[ii-1]
01906
01907                 # Take the values (interpolate)
01908                 K_max = [item[0] for item in curve_fl2_max]
01909                 fl2_max = [item[1] for item in curve_fl2_max]
01910                 K_res_max = np.interp(fl2_ratio, fl2_max, K_max)
01911
01912                 K_min = [item[0] for item in curve_fl2_min]

```

```

01913         fl2_min = [item[1] for item in curve_fl2_min]
01914         K_res_min = np.interp(fl2_ratio, fl2_min, K_min)
01915
01916         # interpolate with distance from fl1 for fl2
01917         # should be logarithmic interpolation but error negligible
01918         K_res = np.interp(fl1_ratio, [fl1_min, fl1_max], [K_res_min, K_res_max])
01919         return K_res
01920
01921
01922     def Concrete01(self):
01923         """
01924         Generate the material model Concrete01 for rectangular section confined concrete (Mander
1988).
01925         See _Concrete01 function for more information. Use this method or Concrete04, not both (only
one material model for ID).
01926         """
01927         _Concrete01(self.IDID, self.eccecc, self.fccfcc, self.eccueccu)
01928         self.InitializedInitialized = True
01929         self.UpdateStoredDataUpdateStoredData()
01930
01931
01932     def Concrete04(self):
01933         """
01934         Generate the material model Concrete04 for rectangular section confined concrete (Mander
1988).
01935         See _Concrete04 function for more information. Use this method or Concrete01, not both (only
one material model for ID).
01936         """
01937         _Concrete04(self.IDID, self.fccfcc, self.eccecc, self.eccueccu, self.EcEc, self.fctfct,
self.etet, self.betabeta)
01938         self.InitializedInitialized = True
01939         self.UpdateStoredDataUpdateStoredData()
01940
01941
01942 class ConfMander1988RectRCRectShape(ConfMander1988Rect):
01943     """
01944     Class that is the children of ConfMander1988Rect and combine the class RCRectShape (section) to
retrieve the information needed.
01945
01946     @param ConfMander1988Rect: Parent class.
01947     """
01948     def __init__(self, ID: int, section: RCRectShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
01949         """
01950         Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
01951         and the section class RCRectShape. wx_bottom, wx_top and wy are computed using the private
method __Compute_w that
01952         and the member variable bars_ranges_position_y and bars_position_x from the section
passed.
01953         The copy of the section passed is stored in the member variable self.sectionsection.
01954
01955         @param ID (int): Unique material model ID.
01956         @param section (RCRectShape): RCRectShape section object.
01957         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
01958         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
01959         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01960         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
01961         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
according to Mander 1988.
01962         @param beta (float, optional): Loading point value defining the exponential curve parameter to
define the residual stress.
Defaults to 0.1 (according to OpenSeesPy documentation)
01963         """
01964
01965         self.sectionsection = deepcopy(section)
01966         ranges = section.bars_ranges_position_y
01967         bars = section.bars_position_x
01968         wy = self.__Compute_w__Compute_w(ranges, section.D_bars)
01969         wx_top = self.__Compute_w__Compute_w(bars[0], section.D_bars)
01970         wx_bottom = self.__Compute_w__Compute_w(bars[-1], section.D_bars)
01971
01972         super().__init__(ID, section.bc, section.dc, section.Ac, section.fc, section.Ec,
section.nr_bars, section.D_bars,
01973         wx_top, wx_bottom, wy, section.s, section.D_hoops, section.rho_s_x, section.rho_s_y,
section.fs,
01974         ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
01975         self.section_name_tagsection_name_tag = section.name_tag
01976         self.UpdateStoredDataUpdateStoredData()
01977
01978     def __Compute_w(self, vector, D_bars):
01979         """
01980         Private method that converts information from the section passed to the format of the class
ConfMander1988Rect.
01981

```

```

01982         @param vector (list): Vector with the information from the section.
01983         @param D_bars (float): Diameter of the bars.
01984
01985         @returns list: Converted information.
01986         """
01987         l = len(vector)
01988         w = np.zeros(l-2)
01989         for i, elem in enumerate(vector[1:l-1]):
01990             w[i] = elem - D_bars
01991         return w
01992
01993
01994 class ConfMander1988Circ(MaterialModels):
01995     """
01996     Class that stores functions and material properties of a RC circular section
01997     with Mander 1988 as the material model for the confined reinforced concrete and the OpenSeesPy
01998     command type used to model it is Concrete04 or Concrete01.
01999     For more information about the empirical model for the computation of the parameters, see Mander
02000     et Al. 1988, Karthik and Mander 2011 and SIA 262:2012.
02001
02002     @param MaterialModels: Parent abstract class.
02003     """
02004     def __init__(self, ID: int, bc, Ac, fc, Ec, nr_bars, D_bars, s, D_hoops, rho_s_vol, fs,
02005                  ec = 1, ecp = 1, fct = -1, et = -1, esu = -1, beta = 0.1):
02006         """
02007         Constructor of the class.
02008
02009         @param ID (int): Unique material model ID.
02010         @param bc (float): Width of the confined core (from the centerline of the hoops, according to
02011         Mander et Al. 1988).
02012         @param Ac (float): Area of the confined core (according to Mander et Al. 1988).
02013         @param fc (float): Compressive concrete yield strength (needs to be negative).
02014         @param Ec (float): Young modulus.
02015         @param nr_bars (float): Number of reinforcement (allow float for computing the equivalent
02016         nr_bars with different reinforcement areas).
02017         @param D_bars (float): Diameter of the vertical reinforcing bars.
02018         @param s (float): Vertical spacing between hoops.
02019         @param D_hoops (float): Diameter of hoops.
02020         @param rho_s_vol (float): Compute the ratio of the volume of transverse confining steel to the
02021         volume of confined concrete core.
02022         @param fs (float): Yield stress for the hoops.
02023         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
02024         according to Karthik and Mander 2011.
02025         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
02026         to Mander 1988.
02027         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02028         according to SIA 262:2012.
02029         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02030         according to SIA 262:2012.
02031         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
02032         according to Mander 1988.
02033         @param beta (float, optional): Loading point value defining the exponential curve parameter to
02034         define the residual stress.
02035         Defaults to 0.1 (according to OpenSeesPy documentation)
02036
02037         @exception NegativeValue: ID needs to be a positive integer.
02038         @exception NegativeValue: bc needs to be positive.
02039         @exception NegativeValue: Ac needs to be positive.
02040         @exception PositiveValue: fc needs to be negative.
02041         @exception NegativeValue: Ec needs to be positive.
02042         @exception NegativeValue: nr_bars needs to be positive.
02043         @exception NegativeValue: D_bars needs to be positive.
02044         @exception NegativeValue: s needs to be positive.
02045         @exception NegativeValue: D_hoops needs to be positive.
02046         @exception NegativeValue: rho_s_vol needs to be positive.
02047         @exception NegativeValue: fs needs to be positive.
02048         @exception PositiveValue: ec needs to be negative if different from 1.
02049         @exception PositiveValue: ecp needs to be negative if different from 1.
02050         @exception NegativeValue: fct needs to be positive if different from -1.
02051         @exception NegativeValue: et needs to be positive if different from -1.
02052         @exception NegativeValue: esu needs to be positive if different from -1.
02053         """
02054         # Check
02055         if ID < 0: raise NegativeValue()
02056         if bc < 0: raise NegativeValue()
02057         if Ac < 0: raise NegativeValue()
02058         if fc > 0: raise PositiveValue()
02059         if Ec < 0: raise NegativeValue()
02060         if nr_bars < 0: raise NegativeValue()
02061         if D_bars < 0: raise NegativeValue()
02062         if s < 0: raise NegativeValue()
02063         if D_hoops < 0: raise NegativeValue()
02064         if rho_s_vol < 0: raise NegativeValue()
02065         if fs < 0: raise NegativeValue()
02066         if ec != 1 and ec > 0: raise PositiveValue()
02067         if ecp != 1 and ecp > 0: raise PositiveValue()
02068         if fct != -1 and fct < 0: raise NegativeValue()

```

```

02058         if et != -1 and et < 0: raise NegativeValue()
02059         if esu != -1 and esu < 0: raise NegativeValue()
02060
02061     # Arguments
02062     self.IDID = ID
02063     self.bcbc = bc
02064     self.AcAc = Ac
02065     self.fcfc = fc
02066     self.EcEc = Ec
02067     self.nr_barsnr_bars = nr_bars
02068     self.D_barsD_bars = D_bars
02069     self.ss = s
02070     self.D_hoopsD_hoops = D_hoops
02071     self.rho_s_volrho_s_vol = rho_s_vol
02072     self.fsfs = fs
02073     self.esuesu = 0.05 if esu == -1 else esu
02074     self.betabeta = beta
02075
02076     # Initialized the parameters that are dependent from others
02077     self.section_name_tagsection_name_tag = "None"
02078     self.InitializedInitialized = False
02079     self.ReInitReInit(ec, ecp, fct, et)
02080
02081     def ReInit(self, ec = 1, ecp = 1, fct = -1, et = -1):
02082         """
02083         Implementation of the homonym abstract method.
02084         See parent class DataManagement for detailed information.
02085
02086         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
02087         according to Karthik and Mander 2011.
02088         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
02089         to Mander 1988.
02090         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02091         according to SIA 262:2012.
02092         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
02093         according to SIA 262:2012.
02094         """
02095         # Check applicability
02096         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
02097
02098     # Arguments
02099     self.ecec = self.Compute_ecCompute_ec() if ec == 1 else ec
02100     self.ecpecp = self.Compute_ecpCompute_ecp() if ecp == 1 else ecp
02101     self.fctfct = self.Compute_fctCompute_fct() if fct == -1 else fct
02102     self.etet = self.Compute_etCompute_et() if et == -1 else et
02103
02104     # Members
02105     s_prime = self.ss - self.D_hoopsD_hoops
02106     self.ecuecu = self.Compute_ecuCompute_ecu()
02107     self.AeAe = math.pi/4 * (self.bcbc - s_prime/2)**2
02108     self.rho_ccrho_cc = self.nr_barsnr_bars*self.D_barsD_bars**2/4.0*math.pi / self.AcAc
02109     self.AccAcc = self.AcAc*(1.0-self.rho_ccrho_cc)
02110     self.keke = self.AeAe/self.AccAcc
02111     self.flfl = -self.rho_s_volrho_s_vol * self.fsfs / 2
02112     self.fl_primefl_prime = self.flfl * self.keke
02113     self.K_comboK_combo = -1.254 + 2.254 * math.sqrt(1.0+7.94*self.fl_primefl_prime/self.fcfc) -
02114     2.0*self.fl_primefl_prime/self.fcfc
02115     self.fccfcc = self.fcfc * self.K_comboK_combo
02116     self.eccecc = self.Compute_eccCompute_ecc()
02117     self.eccueccu = self.Compute_eccuCompute_eccu()
02118     if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
02119     self.section_name_tagsection_name_tag + " (modified)"
02120
02121     # Data storage for loading/saving
02122     self.UpdateStoredDataUpdateStoredData()
02123
02124     # Methods
02125     def UpdateStoredData(self):
02126         """
02127         Implementation of the homonym abstract method.
02128         See parent class DataManagement for detailed information.
02129         """
02130         self.datadata = [{"INFO_TYPE", "ConfMander1988Circ"}, # Tag for differentiating different data
02131         [{"ID", self.IDID},
02132         [{"section_name_tag", self.section_name_tagsection_name_tag},
02133         [{"bc", self.bcbc},
02134         [{"Ac", self.AcAc},
02135         [{"fc", self.fcfc},
02136         [{"Ec", self.EcEc},
02137         [{"ec", self.ecec},
02138         [{"ecp", self.ecpecp},
02139         [{"ecu", self.ecuecu},
02140         [{"fct", self.fctfct},
02141         [{"et", self.etet},
02142         [{"fcc", self.fccfcc},
02143         [{"ecc", self.eccecc},

```

```

02139         ["eccu", self.eccueccu],
02140         ["beta", self.betabeta],
02141         ["nr_bars", self.nr_barsnr_bars],
02142         ["D_bars", self.D_barsD_bars],
02143         ["s", self.ss],
02144         ["D_hoops", self.D_hoopsD_hoops],
02145         ["rho_s_vol", self.rho_s_volrho_s_vol],
02146         ["fs", self.fsfs],
02147         ["esu", self.esuesu],
02148         ["Initialized", self.InitializedInitialized]]
02149
02150
02151     def ShowInfo(self, plot = False, block = False, concrete04 = True):
02152         """
02153         Implementation of the homonym abstract method.
02154         See parent class DataManagement for detailed information.
02155
02156         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
02157         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
02158         @param concrete04 (bool, optional): Option to show in the plot the concrete04 or concrete01 if
False. Defaults to True.
02159         """
02160         print("")
02161         print("Requested info for Confined Mander 1988 (circular) material model Parameters, ID =
{}".format(self.IDID))
02162         print("Section associated: {}".format(self.section_name_tagsection_name_tag))
02163         print('Concrete strength fc = {} MPa'.format(self.fcfc/MPa_unit))
02164         print('Concrete strength confined fcc = {} MPa'.format(self.fccfcc/MPa_unit))
02165         print('Strain at maximal strength ec = {}'.format(self.ecec))
02166         print('Strain at maximal strength confined ecc = {}'.format(self.eccecc))
02167         print('Maximal strain ecu = {}'.format(self.ecuecu))
02168         print('Maximal strain confined eccu = {}'.format(self.eccueccu))
02169         print("")
02170
02171         if plot:
02172             fig, ax = plt.subplots()
02173             if concrete04:
02174                 PlotConcrete04(self.fccfcc, self.EcEc, self.eccecc, self.eccueccu, "C", ax, self.IDID)
02175             else:
02176                 PlotConcrete01(self.fccfcc, self.eccecc, 0.0, self.eccueccu, ax, self.IDID)
02177
02178             if block:
02179                 plt.show()
02180
02181
02182     def CheckApplicability(self):
02183         """
02184         Implementation of the homonym abstract method.
02185         See parent class MaterialModels for detailed information.
02186         """
02187         Check = True
02188         if self.fcfc < -110*MPa_unit: # Deierlein 1999
02189             Check = False
02190             print("With High Strength concrete (< -110 MPa), a better material model should be used
(see Abdesselam et Al. 2019)")
02191             if not Check:
02192                 print("The validity of the equations is not fullfilled.")
02193                 print("!!!!!! WARNING !!!!!!! Check material model of Confined Mander 1988, ID=",
self.IDID)
02194                 print("")
02195
02196
02197     def Compute_ec(self):
02198         """
02199         Method that computes the compressive concrete yield strain.
02200         For more information, see Karthik and Mander 2011.
02201
02202         @returns float: Strain
02203         """
02204         # return -0.002 # Alternative: Mander et Al. 1988
02205         return -0.0015 + self.fcfc/MPa_unit/70000 # Karthik Mander 2011
02206
02207
02208     def Compute_ecp(self):
02209         """
02210         Method that computes the compressive concrete spalling strain.
02211         For more information, see Mander et Al. 1988.
02212
02213         @returns float: Strain
02214         """
02215         return 2.0*self.ecec
02216
02217
02218     def Compute_fct(self):
02219         """

```



```

02220         Method that computes the tensile concrete yield stress.
02221         For more information, see SIA 262:2012. Assume that the confinement do not play an essential
role in tension.
02222
02223         @returns float: Stress.
02224         """
02225         return 0.30 * math.pow(-self.fcfc/MPa_unit, 2/3) * MPa_unit
02226
02227
02228     def Compute_et(self):
02229         """
02230         Method that computes the tensile concrete yield strain.
02231         For more information, see Mander et Al. 1988 (eq 45).
02232
02233         @returns float: Strain.
02234         """
02235         return self.fctfct/self.EcEc
02236
02237
02238     def Compute_ecu(self):
02239         """
02240         Method that computes the compressive concrete failure strain.
02241         For more information, see Karthik and Mander 2011.
02242
02243         @returns float: Strain
02244         """
02245         # return -0.004 # Alternative: Mander et Al. 1988
02246         return -0.012 - 0.0001 * self.fcfc/MPa_unit # Karthik Mander 2011
02247
02248
02249     def Compute_ecc(self):
02250         """
02251         Method that computes the compressive confined concrete yield strain.
02252         For more information, see Karthik and Mander 2011.
02253
02254         @returns float: Strain
02255         """
02256         return (1.0 + 5.0 * (self.K_comboK_combo-1.0)) * self.ecec # Karthik Mander 2011
02257
02258
02259     def Compute_eccu(self):
02260         """
02261         Method that computes the compressive confined concrete failure strain.
02262         For more information, see Karthik and Mander 2011.
02263
02264         @returns float: Strain
02265         """
02266         # return -0.004 + (1.4*(self.rho_s_x+self.rho_s_y)*self.esu*self.fs) / self.fcc # Alternative:
Prof. Katrin Beyer
02267         return 5*self.ecccecc # Karthik Mander 2011
02268
02269
02270     def Concrete01(self):
02271         """
02272         Generate the material model Concrete01 for rectangular section confined concrete (Mander
1988).
02273         See _Concrete01 function for more information. Use this method or Concrete04, not both (only
one material model for ID).
02274         """
02275         _Concrete01(self.IDID, self.ecccecc, self.fccfcc, self.eccueccu)
02276         self.InitializedInitialized = True
02277         self.UpdateStoredDataUpdateStoredData()
02278
02279
02280     def Concrete04(self):
02281         """
02282         Generate the material model Concrete04 for circular section confined concrete (Mander 1988).
02283         See _Concrete04 function for more information. Use this method or Concrete01, not both (only
one material model for ID).
02284         """
02285         _Concrete04(self.IDID, self.fccfcc, self.ecccecc, self.eccueccu, self.EcEc, self.fctfct,
self.etet, self.betabeta)
02286         self.InitializedInitialized = True
02287         self.UpdateStoredDataUpdateStoredData()
02288
02289
02290 class ConfMander1988CircRCCircShape(ConfMander1988Circ):
02291     """
02292     Class that is the children of ConfMander1988Circ and combine the class RCCircShape (section) to
retrieve the information needed.
02293
02294     @param ConfMander1988Circ: Parent class.
02295     """
02296     def __init__(self, ID: int, section: RCCircShape, ec=1, ecp=1, fct=-1, et=-1, esu=-1, beta=0.1):
02297         """
02298         Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class

```

```

02299         and the section class RCCircShape.
02300         The copy of the section passed is stored in the member variable self.sectionsection.
02301
02302         @param ID (int): Unique material model ID.
02303         @param section (RCCircShape): RCCircShape section object.
02304         @param ec (float, optional): Compressive concrete yield strain. Defaults to 1, e.g. computed
according to Karthik and Mander 2011.
02305         @param ecp (float, optional): Concrete spalling strain. Defaults to 1, e.g. computed according
to Mander 1988.
02306         @param fct (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
02307         @param et (float, optional): Tensile concrete yield strain. Defaults to -1, e.g. computed
according to SIA 262:2012.
02308         @param esu (float, optional): Tensile steel bars failure strain. Defaults to -1, e.g. computed
according to Mander 1988.
02309         @param beta (float, optional): Loading point value defining the exponential curve parameter to
define the residual stress.
02310             Defaults to 0.1 (according to OpenSeesPy documentation)
02311         """
02312         self.sectionsection = deepcopy(section)
02313         super().__init__(ID, section.bc, section.Ac, section.fc, section.Ec, section.n_bars,
section.D_bars, section.s, section.D_hoops,
02314             section.rho_s_vol, section.fs, ec=ec, ecp=ecp, fct=fct, et=et, esu=esu, beta=beta)
02315         self.section_name_tagsection_name_tag = section.name_tag
02316         self.UpdateStoredDataUpdateStoredData()
02317
02318
02319 class UniaxialBilinear(MaterialModels):
02320     """
02321     Class that stores functions and material properties of a simple uniaxial bilinear model
02322     with the OpenSeesPy command type used to model it is Steel01.
02323
02324     @param MaterialModels: Parent abstract class.
02325     """
02326     def __init__(self, ID: int, fy, Ey, b = 0.01):
02327         """
02328         Constructor of the class.
02329
02330         @param ID (int): Unique material model ID.
02331         @param fy (float): Yield stress.
02332         @param Ey (float): Young modulus.
02333         @param b (float, optional): Strain hardening factor. Defaults to 0.01.
02334
02335         @exception NegativeValue: ID needs to be a positive integer.
02336         @exception NegativeValue: fy needs to be positive.
02337         @exception NegativeValue: Ey needs to be positive.
02338         """
02339         # Check
02340         if ID < 1: raise NegativeValue()
02341         if fy < 0: raise NegativeValue()
02342         if Ey < 0: raise NegativeValue()
02343
02344         # Arguments
02345         self.IDID = ID
02346         self.fyfy = fy
02347         self.EyEy = Ey
02348         self.bb = b
02349
02350         # Initialized the parameters that are dependent from others
02351         self.section_name_tagsection_name_tag = "None"
02352         self.InitializedInitialized = False
02353         self.ReInitReInit()
02354
02355     def ReInit(self):
02356         """
02357         Implementation of the homonym abstract method.
02358         See parent class DataManagement for detailed information.
02359         """
02360         # Check applicability
02361         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
02362
02363         # Members
02364         self.eyey = self.fyfy / self.EyEy
02365         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
02366
02367         # Data storage for loading/saving
02368         self.UpdateStoredDataUpdateStoredData()
02369
02370
02371     # Methods
02372     def UpdateStoredData(self):
02373         """
02374         Implementation of the homonym abstract method.
02375         See parent class DataManagement for detailed information.
02376         """
02377         self.datadata = [{"INFO_TYPE", "UniaxialBilinear"}], # Tag for differentiating different data

```

```

02378         ["ID", self.IDID],
02379         ["section_name_tag", self.section_name_tagsection_name_tag],
02380         ["fy", self.fyfy],
02381         ["Ey", self.EyEy],
02382         ["ey", self.eyey],
02383         ["b", self.bb],
02384         ["Initialized", self.InitializedInitialized]]
02385
02386
02387     def ShowInfo(self, plot = False, block = False):
02388         """
02389         Implementation of the homonym abstract method.
02390         See parent class DataManagement for detailed information.
02391
02392         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
02393         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
02394         """
02395         print("")
02396         print("Requested info for Uniaxial Bilinear material model Parameters, ID =
{}".format(self.IDID))
02397         print("Section associated: {}".format(self.section_name_tagsection_name_tag))
02398         print('Yielding stress fy = {} MPa'.format(self.fyfy/MPa_unit))
02399         print('Young modulus Ey = {} MPa'.format(self.EyEy/MPa_unit))
02400         print('Maximal elastic strain epsilon y = {}'.format(self.eyey))
02401         print('Hardening factor b = {}'.format(self.bb))
02402         print("")
02403
02404         if plot:
02405             # Data for plotting
02406             e_pl = 10.0 * self.eyey # to show that if continues with this slope
02407             sigma_pl = self.bb * self.EyEy * e_pl
02408
02409             x_axis = np.array([0.0, self.eyey, (self.eyey+e_pl)])*100
02410             y_axis = np.array([0.0, self.fyfy, (self.fyfy+sigma_pl)])/MPa_unit
02411
02412             fig, ax = plt.subplots()
02413             ax.plot(x_axis, y_axis, 'k-')
02414
02415             ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
02416                   title='Uniaxial Bilinear model for material ID={}'.format(self.IDID))
02417             ax.grid()
02418
02419             if block:
02420                 plt.show()
02421
02422
02423     def CheckApplicability(self):
02424         """
02425         Implementation of the homonym abstract method.
02426         See parent class MaterialModels for detailed information.
02427         """
02428         Check = True
02429         # if len(self.wy) == 0 or len(self.wx_top) == 0 or len(self.wx_bottom) == 0:
02430         #     Check = False
02431         #     print("Hypothesis of one bar per corner not fullfilled.")
02432         if not Check:
02433             print("The validity of the equations is not fullfilled.")
02434             print("!!!!!!! WARNING !!!!!!! Check material model of Uniaxial Bilinear, ID=", self.IDID)
02435             print("")
02436
02437
02438     def Steel01(self):
02439         """
02440         Generate the material model Steel01 uniaxial bilinear material model.
02441         See _Steel01 function for more information.
02442         """
02443         _Steel01(self.IDID, self.fyfy, self.EyEy, self.bb)
02444         self.InitializedInitialized = True
02445         self.UpdateStoredDataUpdateStoredData()
02446
02447
02448 class UniaxialBilinearSteelIShape(UniaxialBilinear):
02449     """
02450     Class that is the children of UniaxialBilinear and combine the class SteelIShape (section) to
retrieve the information needed.
02451
02452     @param UniaxialBilinear: Parent class.
02453     """
02454     def __init__(self, ID: int, section: SteelIShape, b=0.01):
02455         """
02456         Constructor of the class. It passes the arguments into the parent class to generate the
combination of the parent class
and the section class SteelIShape.
02457         The copy of the section passed is stored in the member variable self.sectionsection.
02458
02459

```

```

02460         @param ID (int): Unique material model ID.
02461         @param section (SteelIShape): SteelIShape section object.
02462         @param b (float, optional): Strain hardening factor. Defaults to 0.01.
02463         """
02464         self.sectionsection = deepcopy(section)
02465         super().__init__(ID, section.Fy, section.E, b=b)
02466         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
02467         self.UpdateStoredDataUpdateStoredData()
02468
02469
02470 class GMP1970(MaterialModels):
02471     """
02472     Class that stores functions and material properties of the vertical steel reinforcement bars
02473     with Giuffré, Menegotto and Pinto 1970 as the material model and the OpenSeesPy command type
02474     used to model it is Steel02.
02475     For more information about the empirical model for the computation of the parameters, see Giuffré,
02476     Menegotto and Pinto 1970 and Carreno et Al. 2020.
02477
02478     @param MaterialModels: Parent abstract class.
02479     """
02480     def __init__(self, ID: int, fy, Ey, b = 0.02, R0 = 20, cR1 = 0.9, cR2 = 0.08, a1 = 0.039, a2 =
02481         1.0, a3 = 0.029, a4 = 1.0):
02482         """
02483         Constructor of the class. The parameters are suggested as exposed in Carreno et Al. 2020 but
02484         also the one suggested by OpenSeesPy documentation are reliable
02485         (b = 0.015, R0 = 10, cR1 = 0.925, cR2 = 0.15).
02486
02487         @param ID (int): Unique material model ID.
02488         @param fy (float): Steel yield strength.
02489         @param Ey (float): Young modulus.
02490         @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et
02491         Al. 2020.
02492         @param R0 (int, optional): First parameter to control the transition from elastic to plastic
02493         branches. Defaults to 20, according to Carreno et Al. 2020.
02494         @param cR1 (float, optional): Second parameter to control the transition from elastic to
02495         plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
02496         @param cR2 (float, optional): Third parameter to control the transition from elastic to
02497         plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
02498         @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield
02499         envelope as proportion of yield strength after a plastic strain.
02500         Defaults to 0.039, according to Carreno et Al. 2020.
02501         @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al.
02502         2020.
02503         @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope
02504         as proportion of yield strength after a plastic strain.
02505         Defaults to 0.029, according to Carreno et Al. 2020.
02506         @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al.
02507         2020.
02508
02509         @exception NegativeValue: ID needs to be a positive integer.
02510         """
02511         # Check
02512         if ID < 1: raise NegativeValue()
02513
02514         # Arguments
02515         self.IDID = ID
02516         self.fyfy = fy
02517         self.EyEy = Ey
02518         self.bb = b
02519         self.R0R0 = R0
02520         self.cR1cR1 = cR1
02521         self.cR2cR2 = cR2
02522         self.a1a1 = a1
02523         self.a2a2 = a2
02524         self.a3a3 = a3
02525         self.a4a4 = a4
02526
02527         # Initialized the parameters that are dependent from others
02528         self.section_name_tagsection_name_tag = "None"
02529         self.InitializedInitialized = False
02530         self.ReInitReInit()
02531
02532     def ReInit(self):
02533         """
02534         Implementation of the homonym abstract method.
02535         See parent class DataManagement for detailed information.
02536         """
02537         # Check applicability
02538         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
02539
02540         # Members
02541         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
02542         self.section_name_tagsection_name_tag + " (modified)"
02543
02544         # Data storage for loading/saving
02545         self.UpdateStoredDataUpdateStoredData()
02546
02547

```

```

02534
02535 # Methods
02536 def UpdateStoredData(self):
02537     """
02538     Implementation of the homonym abstract method.
02539     See parent class DataManagement for detailed information.
02540     """
02541     self.data = [{"INFO_TYPE": "GMP1970"}, # Tag for differentiating different data
02542                  [{"ID": self.IDID},
02543                   [{"section_name_tag": self.section_name_tag, "section_name_tag": self.section_name_tag},
02544                    [{"fy": self.fyfy},
02545                     [{"Ey": self.EyEy},
02546                      [{"b": self.bb},
02547                       [{"R0": self.R0R0},
02548                        [{"cR1": self.cR1cR1},
02549                         [{"cR2": self.cR2cR2},
02550                          [{"a1": self.a1a1},
02551                           [{"a2": self.a2a2},
02552                            [{"a3": self.a3a3},
02553                             [{"a4": self.a4a4},
02554                              [{"Initialized": self.Initialized}]]]]]]]]]]]]]]]
02555
02556
02557 def ShowInfo(self):
02558     """
02559     Implementation of the homonym abstract method.
02560     See parent class DataManagement for detailed information.
02561     """
02562     print("")
02563     print("Requested info for GMP1970 (Giuffr -Menegotto-Pinto) material model Parameters, ID = {}".format(self.IDID))
02564     print("Section associated: {}".format(self.section_name_tag))
02565     print("Yield stress fy = {} MPa".format(self.fyfy/MPa_unit))
02566     print("Young modulus Ey = {} MPa".format(self.EyEy/MPa_unit))
02567     print("Strain hardening ratio b = {}".format(self.bb))
02568     print("Bauschinger effect factors R0 = {}, cR1 = {} and cR2 = {}".format(self.R0R0, self.cR1cR1, self.cR2cR2))
02569     print("Isotropic hardening factors a1 = {}, a2 = {}, a3 = {} and a4 = {}".format(self.a1a1, self.a2a2, self.a3a3, self.a4a4))
02570     print("")
02571
02572     #TODO: add plot option (difficult to implement)
02573
02574
02575 def CheckApplicability(self):
02576     """
02577     Implementation of the homonym abstract method.
02578     See parent class MaterialModels for detailed information.
02579     """
02580     Check = True
02581     # No checks
02582     if not Check:
02583         print("The validity of the equations is not fulfilled.")
02584         print("!!!!!! WARNING !!!!!!! Check material model of GMP1970, ID=", self.IDID)
02585         print("")
02586
02587
02588 def Steel02(self):
02589     """
02590     Generate the material model Steel02 uniaxial Giuffr -Menegotto-Pinto steel material with isotropic strain hardening.
02591     See _Steel02 function for more information.
02592     """
02593     _Steel02(self.IDID, self.fyfy, self.EyEy, self.bb, self.R0R0, self.cR1cR1, self.cR2cR2, self.a1a1, self.a2a2, self.a3a3, self.a4a4)
02594     self.Initialized = True
02595     self.UpdateStoredData()
02596
02597
02598 class GMP1970RCRectShape(GMP1970):
02599     """
02600     Class that is the children of GMP1970 and combine the class RCRectShape (section) to retrieve the information needed.
02601
02602     @param GMP1970: Parent class.
02603     """
02604     def __init__(self, ID: int, section: RCRectShape, b=0.02, R0=20.0, cR1=0.9, cR2=0.08, a1=0.039, a2=1.0, a3=0.029, a4=1.0):
02605         """
02606         Constructor of the class. It passes the arguments into the parent class to generate the combination of the parent class and the section class RCRectShape.
02607         and the section class RCRectShape.
02608         The copy of the section passed is stored in the member variable self.section.
02609
02610         @param ID (int): Unique material model ID.
02611         @param section (RCRectShape): RCRectShape section object.
02612         @param b (float, optional): Strain-hardening ratio. Defaults to 0.02, according to Carreno et

```

```

Al. 2020.
02613     @param R0 (int, optional): First parameter to control the transition from elastic to plastic
branches. Defaults to 20, according to Carreno et Al. 2020.
02614     @param cR1 (float, optional): Second parameter to control the transition from elastic to
plastic branches. Defaults to 0.9, according to Carreno et Al. 2020.
02615     @param cR2 (float, optional): Third parameter to control the transition from elastic to
plastic branches. Defaults to 0.08, according to Carreno et Al. 2020.
02616     @param a1 (float, optional): Isotropic hardening parameter, increase of compression yield
envelope as proportion of yield strength after a plastic strain.
02617         Defaults to 0.039, according to Carreno et Al. 2020.
02618     @param a2 (float, optional): Coupled with a1. Defaults to 1.0, according to Carreno et Al.
2020.
02619     @param a3 (float, optional): Isotropic hardening parameter, increase of tension yield envelope
as proportion of yield strength after a plastic strain.
02620         Defaults to 0.029, according to Carreno et Al. 2020.
02621     @param a4 (float, optional): Coupled with a3. Defaults to 1.0, according to Carreno et Al.
2020.
02622     """
02623     self.sectionsection = deepcopy(section)
02624     super().__init__(ID, section.fy, section.Ey, b=b, R0=R0, cR1=cR1, cR2=cR2, a1=a1, a2=a2,
a3=a3, a4=a4)
02625     self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
02626     self.UpdateStoredDataUpdateStoredData()
02627
02628
02629 class UVC(MaterialModels):
02630     """
02631     Class that stores functions and material properties of a steel profile or reinforcing bar
02632     with Updated Voce-Chaboche as the material model and the OpenSeesPy command type used to model
it is UVCuniaxial.
02633     For more information about the how to calibrate the set of parameters, see
02634     de Castro e Sousa, Suzuki and Lignos 2020 and Hartloper, de Castro e Sousa and Lignos 2021.
02635
02636     @param MaterialModels: Parent abstract class.
02637     """
02638     def __init__(self, ID: int, fy, Ey, QInf, b, DInf, a, cK: np.ndarray, gammaK: np.ndarray):
02639         """
02640         Constructor of the class.
02641
02642         @param ID (int): Unique material model ID.
02643         @param fy (float): Initial yield stress of the steel material.
02644         @param Ey (float): Elastic modulus of the steel material.
02645         @param QInf (float): Maximum increase in yield stress due to cyclic hardening (isotropic
hardening).
02646         @param b (float): Saturation rate of QInf.
02647         @param DInf (float): Decrease in the initial yield stress, to neglect the model updates set
DInf = 0.
02648         @param a (float): Saturation rate of DInf, a > 0. If DInf == 0, then a is arbitrary (but still
a > 0).
02649         @param cK (np.ndarray): Array of 1 dimension; each entry is one kinematic hardening parameter
associated with one backstress, up to 8 may be specified.
02650         @param gammaK (np.ndarray): Array of 1 dimension; each entry is one saturation rate of
kinematic hardening associated with one backstress, up to 8 may be specified.
02651
02652         @exception NegativeValue: ID needs to be a positive integer.
02653         @exception NegativeValue: fy needs to be positive.
02654         @exception NegativeValue: Ey needs to be positive.
02655         @exception NegativeValue: QInf needs to be positive.
02656         @exception NegativeValue: b needs to be positive.
02657         @exception NegativeValue: DInf needs to be positive.
02658         @exception NegativeValue: a needs to be positive.
02659         @exception WrongArgument: cK can't be empty.
02660         @exception WrongArgument: cK and gammaK have as many entries as the number of backstresses
(thus they have the same length).
02661         """
02662         # Check
02663         if ID < 1: raise NegativeValue()
02664         if fy < 0: raise NegativeValue()
02665         if Ey < 0: raise NegativeValue()
02666         if QInf < 0: raise NegativeValue()
02667         if b < 0: raise NegativeValue()
02668         if DInf < 0: raise NegativeValue()
02669         if a < 0: raise NegativeValue()
02670         if len(cK) == 0: raise WrongArgument()
02671         if len(cK) != len(gammaK): raise WrongArgument()
02672         if len(cK) != 2: print("!!!!!! WARNING !!!!!!! Number of backstresses should be 2 for optimal
performances")
02673         if DInf == 0: print("!!!!!! WARNING !!!!!!! With DInf = 0, the model used is Voce-Chaboche
(VC) not updated (UVC)")
02674
02675         # Arguments
02676         self.IDID = ID
02677         self.fyfy = fy
02678         self.EyEy = Ey
02679         self.QInfQInf = QInf
02680         self.bb = b
02681         self.DInfDInf = DInf

```

```

02682         self.aa = a
02683         self.cKcK = copy(cK)
02684         self.gammaKgammaK = copy(gammaK)
02685
02686         # Initialized the parameters that are dependent from others
02687         self.section_name_tagsection_name_tag = "None"
02688         self.InitializedInitialized = False
02689         self.ReInitReInit()
02690
02691     def ReInit(self):
02692         """
02693         Implementation of the homonym abstract method.
02694         See parent class DataManagement for detailed information.
02695         """
02696         # Check applicability
02697         self.CheckApplicabilityCheckApplicabilityCheckApplicability()
02698
02699         # Members
02700         self.NN = len(self.cKcK)
02701         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
02702
02703         # Data storage for loading/saving
02704         self.UpdateStoredDataUpdateStoredData()
02705
02706     # Methods
02707     def UpdateStoredData(self):
02708         """
02709         Implementation of the homonym abstract method.
02710         See parent class DataManagement for detailed information.
02711         """
02712         self.datadata = [{"INFO_TYPE", "UVC"}, # Tag for differentiating different data
02713         [{"ID", self.IDID},
02714         [{"section_name_tag", self.section_name_tagsection_name_tag},
02715         [{"fy", self.fyfy},
02716         [{"Ey", self.EyEy},
02717         [{"QInf", self.QInfQInf},
02718         [{"b", self.bb},
02719         [{"DInf", self.DInfDInf},
02720         [{"a", self.aa},
02721         [{"N", self.NN},
02722         [{"ck", self.cKcK},
02723         [{"gammaK", self.gammaKgammaK},
02724         [{"Initialized", self.InitializedInitialized}]]
02725
02726     def ShowInfo(self):
02727         """
02728         Implementation of the homonym abstract method.
02729         See parent class DataManagement for detailed information.
02730         """
02731         print("")
02732         print("Requested info for UVC material model Parameters, ID = {}".format(self.IDID))
02733         print("Section associated: {} ".format(self.section_name_tagsection_name_tag))
02734         print("Yield strength fy = {} MPa".format(self.fyfy/MPa_unit))
02735         print("Young modulus Ey = {} MPa".format(self.EyEy/MPa_unit))
02736         print("Isotropic hardening factor QInf = {} MPa and saturation rate b =
02737         {}".format(self.QInfQInf/MPa_unit, self.bb))
02738         print("Decrease the initial yield stress DInf = {} MPa and saturation rate a =
02739         {}".format(self.DInfDInf/MPa_unit, self.aa))
02740         print("Kinematic hardening vector ({} backstresses) cK = {} MPa".format(self.NN,
self.cKcK/MPa_unit))
02741         print("And associated saturation rate gammaK = {}".format(self.gammaKgammaK))
02742         print("")
02743
02744         #TODO: implement plot (too complex for now)
02745
02746     def CheckApplicability(self):
02747         """
02748         Implementation of the homonym abstract method.
02749         See parent class MaterialModels for detailed information.
02750         """
02751         Check = True
02752         # No checks
02753         if not Check:
02754             print("The validity of the equations is not fullfilled.")
02755             print("!!!!!! WARNING !!!!!!! Check material model of UVC, ID=", self.IDID)
02756             print("")
02757
02758     def UVCuniaxial(self):
02759         """
02760         Generate the material model Updated Voce-Chaboche (UVC) for uniaxial stress states.
02761         See _UVCuniaxial function for more information.
02762         """
02763
02764

```

```

02765         _UVCuniaxial(self.IDID, self.EyEy, self.fyfy, self.QInfQInf, self.bb, self.DInfDInf, self.aa,
02766                     self.NN, self.CKcK, self.gammaKgammaK)
02767         self.InitializedInitialized = True
02768         self.UpdateStoredDataUpdateStoredData()
02769
02770 class UVCCalibrated(UVC):
02771     """
02772     Class that is the children of UVC that retrieve calibrated parameters from
02773     UVC_calibrated_parameters.txt.
02774     The file text can be modified by adding more calibrated parameters.
02775
02776     @param UVC: Parent class.
02777     """
02778     def __init__(self, ID: int, calibration: str, fy = -1, E = -1):
02779         """
02780         Constructor of the class. It retrieve the parameters from UVC_calibrated_parameters.txt and
02781         pass them in the parent class.
02782
02783         @param ID (int): Unique material model ID.
02784         @param calibration (str): Label of the calibration parameter set. The options are: \n
02785         # 'S355J2_25mm_plate' \n
02786         # 'S355J2_50mm_plate' \n
02787         # 'S355J2_HEB500_flange' \n
02788         # 'S355J2_HEB500_web' \n
02789         # 'S460NL_25mm_plate' \n
02790         # 'S690QL_25mm_plate' \n
02791         # 'A992Gr50_W14X82_web' \n
02792         # 'A992Gr50_W14X82_flange' \n
02793         # 'A500GrB_HSS305X16' \n
02794         # 'BCP325_22mm_plate' \n
02795         # 'BCR295_HSS350X22' \n
02796         # 'HYP400_27mm_plate' \n
02797         @param fy (float, optional): Yield strength. Defaults to -1, e.g. taken equal to the one given
02798         in the calibration parameter set.
02799         @param E (float, optional): Young modulus. Defaults to -1, e.g. taken equal to the one given
02800         in the calibration parameter set.
02801
02802         @exception NegativeValue: fy needs to be positive if different from -1.
02803         @exception NegativeValue: E needs to be positive if different from -1.
02804         @exception NameError: calibration needs to be equal to the label of one of the set of
02805         calibrated parameters.
02806         """
02807         if fy != -1 and fy < 0: raise NegativeValue()
02808         if E != -1 and E < 0: raise NegativeValue()
02809
02810         self.calibrationcalibration = calibration
02811
02812         # Structure of the data to be stored
02813         names = ["Material", "Ey", "fy", "QInf", "b", "DInf", "a", "C1", "gamma1", "C2", "gamma2"]
02814         # Get the data
02815         __location__ = os.path.realpath(os.path.join(os.getcwd(), os.path.dirname(__file__)))
02816         UVC_data = np.genfromtxt(os.path.join(__location__, 'UVC_calibrated_parameters.txt'),
02817                                dtype=None, skip_header=1, names = names, encoding='ascii', delimiter='\t')
02818         # Define the index (with the location of the correct set of parameters)
02819         index = UVC_data["Material"] == calibration
02820         fy = UVC_data["fy"][index][0]*MPa_unit if fy == -1 else fy
02821         E = UVC_data["Ey"][index][0]*GPa_unit if E == -1 else E
02822         # Check
02823         if not index.any(): raise NameError("No calibrated parameters with that name. Note that there
02824         are no spaces in the label.")
02825
02826         # Assign arguments value
02827         super().__init__(ID, fy, E, UVC_data["QInf"][index][0]*MPa_unit, UVC_data["b"][index][0],
02828                         UVC_data["DInf"][index][0]*MPa_unit, UVC_data["a"][index][0],
02829                         np.array([UVC_data["C1"][index][0], UVC_data["C2"][index][0]])*MPa_unit,
02830                         np.array([UVC_data["gamma1"][index][0], UVC_data["gamma2"][index][0]]))
02831
02832 class UVCCalibratedRCRectShape(UVCCalibrated):
02833     """
02834     Class that is the children of UVCCalibrated and combines the class RCRectShape (section) to
02835     retrieve the information needed.
02836
02837     @param UVCCalibrated: Parent class.
02838     """
02839     def __init__(self, ID: int, section: RCRectShape, calibration = 'S460NL_25mm_plate'):
02840         """
02841         Constructor of the class.
02842
02843         @param ID (int): Unique material model ID.
02844         @param section (RCRectShape): RCRectShape section object.
02845         @param calibration (str): Label of the calibration parameter set. The options are listed in
02846         UVCCalibrated.
02847         Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material
02848         properties.
02849         """

```



```

02841         self.sectionsection = deepcopy(section)
02842         super().__init__(ID, calibration, section.fy, section.Ey)
02843         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
02844         self.UpdateStoredDataUpdateStoredData()
02845
02846
02847 class UVCCalibratedRCCircShape(UVCCalibrated):
02848     """
02849     Class that is the children of UVCCalibrated and combine the class RCCircShape (section) to
    retrieve the information needed.
02850
02851     @param UVCCalibrated: Parent class.
02852     """
02853     def __init__(self, ID: int, section: RCCircShape, calibration = 'S460NL_25mm_plate'):
02854         """
02855         Constructor of the class.
02856
02857         @param ID (int): Unique material model ID.
02858         @param section (RCCircShape): RCCircShape section object.
02859         @param calibration (str, optional): Label of the calibration parameter set. The options are
    listed in UVCCalibrated.
02860         Defaults to 'S460NL_25mm_plate'. Change it accordingly to the steel rebars material
    properties.
02861         """
02862         self.sectionsection = deepcopy(section)
02863         super().__init__(ID, calibration, section.fy, section.Ey)
02864         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
02865         self.UpdateStoredDataUpdateStoredData()
02866
02867
02868 class UVCCalibratedSteelIShapeFlange(UVCCalibrated):
02869     """
02870     Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to
    retrieve the information needed
02871     for the material model of the flange (often used fo the entire section).
02872
02873     @param UVCCalibrated: Parent class.
02874     """
02875     def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_flange'):
02876         """
02877         Constructor of the class.
02878
02879         @param ID (int): Unique material model ID.
02880         @param section (SteelIShape): SteelIShape section object.
02881         @param calibration (str, optional): Label of the calibration parameter set. The options are
    listed in UVCCalibrated.
02882         Defaults to 'S355J2_HEB500_flange'. Change it accordingly to the steel rebars material
    properties.
02883         """
02884         self.sectionsection = deepcopy(section)
02885         super().__init__(ID, calibration, section.Fy, section.E)
02886         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
02887         self.UpdateStoredDataUpdateStoredData()
02888
02889
02890 class UVCCalibratedSteelIShapeWeb(UVCCalibrated):
02891     """
02892     Class that is the children of UVCCalibrated and combine the class SteelIShape (section) to
    retrieve the information needed
02893     for the material model of the web.
02894
02895     @param UVCCalibrated: Parent class.
02896     """
02897     def __init__(self, ID: int, section: SteelIShape, calibration = 'S355J2_HEB500_web'):
02898         """
02899         Constructor of the class.
02900
02901         @param ID (int): Unique material model ID.
02902         @param section (SteelIShape): SteelIShape section object.
02903         @param calibration (str, optional): Label of the calibration parameter set. The options are
    listed in UVCCalibrated.
02904         Defaults to 'S355J2_HEB500_web'. Change it accordingly to the steel rebars material
    properties.
02905         """
02906         self.sectionsection = deepcopy(section)
02907         super().__init__(ID, calibration, section.Fy_web, section.E)
02908         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
02909         self.UpdateStoredDataUpdateStoredData()
02910
02911
02912 # Public functions
02913 def Concrete04Func(fc, discretized_eps, ec, Ec):
02914     """
02915     Function with the equation of the curve of the confined and unconfined concrete (Popovics 1973).
02916
02917     @param fc (float): Compressive concrete yield stress (negative).
02918     @param discretized_eps (float): Variable strain.

```

```

02919     @param ec (float): Compressive concrete yield strain (negative).
02920     @param Ec (float): Concrete Young modulus.
02921
02922     @returns float: Stress in function of variable strain.
02923     """
02924     x = discretized_eps/ec
02925     r = Ec / (Ec - fc/ec)
02926     return fc*x*r / (r-1+x*r)
02927
02928
02929 def PlotConcrete04(fc, Ec, ec, ecu, Type: str, ax, ID = 0):
02930     """
02931     Function that plots the confined/unconfined Concrete04 stress-strain curve.
02932
02933     @param fc (float): Compressive concrete yield strength (needs to be negative).
02934     @param Ec (float): Young modulus.
02935     @param ec (float): Compressive concrete yield strain.
02936     @param ecu (float): Compressive concrete failure strain (negative).
02937     @param Type (str): Type of concrete (confined = 'C', unconfined = 'U')
02938     @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
02939     @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).
02940
02941     @exception NameError:
02942
02943     Example: to create the plot, call this line to pass the correct ax:
02944     fig, ax = plt.subplots()
02945     """
02946     if Type == "C":
02947         name = "Confined (Co04)"
02948     elif Type == "U":
02949         name = "Unconfined (Co04)"
02950     else:
02951         raise NameError("Type should be C or U (ID={})".format(ID))
02952
02953     # Data for plotting
02954     N = 1000
02955     x_axis = np.zeros(N)
02956     y_axis = np.zeros(N)
02957     for i in range(N):
02958         x_axis[i] = i/N*ecu
02959         y_axis[i] = Concrete04Funcnt(fc, x_axis[i], ec, Ec)
02960
02961
02962     ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k-', label = name)
02963     ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
02964           title='Mander 1988 (Concrete04) material model (ID={})'.format(ID))
02965     plt.legend()
02966     plt.grid()
02967
02968
02969 def Concrete01Funcnt(fc, ec, fpcu, ecu, discretized_eps):
02970     """
02971     Function with the equation of the curve of the Concrete01 model.
02972     For more information, see Kent-Scott-Park concrete material object with
02973     degraded linear unloading/reloading stiffness according to the work of Karsan-Jirsa and no
02974     tensile strength.
02975
02976     @param fc (float): Compressive concrete yield stress (negative).
02977     @param ec (float): Compressive concrete yield strain (negative).
02978     @param fpcu (float): Concrete crushing strength (negative).
02979     @param ecu (float): Concrete strain at crushing strength (negative).
02980     @param discretized_eps (float): Variable strain.
02981
02982     @returns float: Stress in function of variable strain.
02983     """
02984     if discretized_eps > ec:
02985         eta = discretized_eps/ec;
02986         return fc*(2*eta-eta*eta);
02987     else:
02988         Ttangent = (fc-fpcu)/(ec-ecu)
02989         return fc + Ttangent*(discretized_eps-ec);
02990
02991 def PlotConcrete01(fc, ec, fpcu, ecu, ax, ID = 0):
02992     """
02993     Function that plots the Concrete01 stress-strain curve.
02994
02995     @param fc (float): Compressive concrete yield stress (negative).
02996     @param ec (float): Compressive concrete yield strain (negative).
02997     @param fpcu (float): Concrete crushing strength (negative).
02998     @param ecu (float): Concrete strain at crushing strength (negative).
02999     @param ax (matplotlib.axes._subplots.AxesSubplot): The figure's wrapper.
03000     @param ID (int, optional): ID of the material model. Defaults to 0 (= not defined).
03001
03002     Example: to create the plot, call this line to pass the correct ax:
03003     fig, ax = plt.subplots()
03004     """

```

```

03005
03006     # Data for plotting
03007     N = 1000
03008     x_axis = np.zeros(N)
03009     y_axis = np.zeros(N)
03010     for i in range(N):
03011         x_axis[i] = i/N*ecu
03012         y_axis[i] = Concrete01Funcnt(fc, ec, fpcu, ecu, x_axis[i])
03013
03014
03015     ax.plot(x_axis*100.0, y_axis/MPa_unit, 'k--', label = "Co01")
03016     ax.set(xlabel='Strain [%]', ylabel='Stress [MPa]',
03017           title='Mander 1988 (Concrete01) material model (ID={})'.format(ID))
03018     plt.legend()
03019     plt.grid()
03020
03021
03022 # Private functions
03023 def _Bilin(ID, Ke, a_s, My_star, theta_p, theta_pc, K, theta_u, rate_det):
03024     """
03025     Private function that generates the material model Bilin.
03026     OpenSeesPy command: \n
03027     uniaxialMaterial("Bilin", IDMat, K, asPos, asNeg, MyPos, MyNeg, LS, LK, LA, LD, cS, cK, cA, cD,
03028     th_pP, th_pN, th_pcP, th_pcN, ResP, ResN, th_uP, th_uN, DP, DN) \n
03029     Parameters (see OpenSeesPy documentation for more information): \n
03030     ID          Material Identification (integer)
03031     K           Initial stiffness after the modification for n (see Ibarra and Krawinkler, 2005)
03032     asPos       Strain hardening ratio after n modification (see Ibarra and Krawinkler, 2005)
03033     asNeg       Strain hardening ratio after n modification (see Ibarra and Krawinkler, 2005)
03034     MyPos       Positive yield moment (with sign)
03035     MyNeg       Negative yield moment (with sign)
03036     LS = 1000   Basic strength deterioration parameter (see Lignos and Krawinkler, 2009) (a very large
03037     # = no cyclic deterioration)
03038     LK = 1000   Unloading stiffness deterioration parameter (see Lignos and Krawinkler, 2009) (a very
03039     large # = no cyclic deterioration)
03040     LA = 1000   Accelerated reloading stiffness deterioration parameter (see Lignos and Krawinkler,
03041     2009) (a very large # = no cyclic deterioration)
03042     LD = 1000   Post-capping strength deterioration parameter (see Lignos and Krawinkler, 2009) (a very
03043     large # = no cyclic deterioration)
03044     cS = 1      Exponent for basic strength deterioration (c = 1.0 for no deterioration)
03045     cK = 1      Exponent for unloading stiffness deterioration (c = 1.0 for no deterioration)
03046     cA = 1      Exponent for accelerated reloading stiffness deterioration (c = 1.0 for no
03047     deterioration)
03048     cD = 1      Exponent for post-capping strength deterioration (c = 1.0 for no deterioration)
03049     th_pP       Plastic rotation capacity for positive loading direction (exemple 0.025)
03050     th_pN       Plastic rotation capacity for negative loading direction (exemple 0.025)
03051     th_pcP      Post-capping rotation capacity for positive loading direction (exemple 0.3)
03052     th_pcN      Post-capping rotation capacity for negative loading direction (exemple 0.3)
03053     KP          Residual strength ratio for positive loading direction (exemple 0.4)
03054     KN          Residual strength ratio for negative loading direction (exemple 0.4)
03055     th_uP       Ultimate rotation capacity for positive loading direction (exemple 0.4)
03056     th_uN       Ultimate rotation capacity for negative loading direction (exemple 0.4)
03057     rateDetP    Rate of cyclic deterioration for positive loading direction (exemple 1.0)
03058     rateDetN    Rate of cyclic deterioration for negative loading direction (exemple 1.0)
03059     """
03060     uniaxialMaterial("Bilin", ID, Ke, a_s, a_s, My_star, -1.0*My_star, 1., 1., 1., 1., 1., 1., 1., 1.,
03061     theta_p, theta_p, theta_pc, theta_pc,
03062     K, K, theta_u, theta_u, rate_det, rate_det)
03063
03064
03065 def _Hysteretic(ID, M1, gamma1, M2, gamma2, M3, gamma3, pinchx, pinchy, dmg1, dmg2, beta):
03066     """
03067     Private function that generates the material model Hysteretic.
03068     OpenSeesPy command: \n
03069     uniaxialMaterial('Hysteretic', matTag, *p1, *p2, *p3=p2, *n1, *n2, *n3=n2, pinchX, pinchY,
03070     damage1, damage2, beta=0.0) \n
03071     Parameters (see OpenSeesPy documentation for more information): \n
03072     matTag      integer tag identifying material
03073     p1          stress and strain (or force & deformation) at first point of the envelope in the
03074     positive direction
03075     p2          stress and strain (or force & deformation) at second point of the envelope in the
03076     positive direction
03077     p3          stress and strain (or force & deformation) at third point of the envelope in the
03078     positive direction (optional)
03079     n1          stress and strain (or force & deformation) at first point of the envelope in the
03080     negative direction
03081     n2          stress and strain (or force & deformation) at second point of the envelope in the
03082     negative direction
03083     n3          stress and strain (or force & deformation) at third point of the envelope in the
03084     negative direction (optional)
03085     pinchX      pinching factor for strain (or deformation) during reloading
03086     pinchY      pinching factor for stress (or force) during reloading
03087     damage1     damage due to ductility: D1(mu-1)
03088     damage2     damage due to energy: D2(Eii/Eult)
03089     beta        power used to determine the degraded unloading stiffness based on ductility, mu-beta
03090     (optional, default=0.0)
03091     """
03092

```

```

03077     uniaxialMaterial("Hysteretic", ID, M1, gamma1, M2, gamma2, M3, gamma3, -M1, -gamma1, -M2, -gamma2,
03078     -M3, -gamma3,
03079     pinchx, pinchy, dmg1, dmg2, beta)
03080
03081 def _Concrete04(ID, fc, ec, ecu, Ec, fct, et, beta):
03082     """
03083     Private function that generates the material model Concrete04 Popovics Concrete material model.
03084     OpenSeesPy command: \n
03085     uniaxialMaterial("Concrete04", matTag, fc, ec, ecu, Ec, <fct et> <beta>) \n
03086     Parameters (see OpenSeesPy documentation for more information): \n
03087     matTag      integer tag identifying material
03088     fc          floating point values defining concrete compressive strength at 28 days (compression is
03089     negative)*
03089     ec          floating point values defining concrete strain at maximum strength*
03090     ecu         floating point values defining concrete strain at crushing strength*
03091     Ec          floating point values defining initial stiffness**
03092     fct         floating point value defining the maximum tensile strength of concrete
03093     et          floating point value defining ultimate tensile strain of concrete
03094     beta        floating point value defining the exponential curve parameter to define the residual stress
03095     (as a factor of ft) at etu
03096     """
03096     uniaxialMaterial("Concrete04", ID, fc, ec, ecu, Ec, fct, et, beta)
03097
03098
03099 def _Concrete01(ID, ec, fc, ecu, fpcu = 0.0):
03100     """
03101     Private function that generates the material model Concrete02 concrete material model.
03102     OpenSeesPy command: \n
03103     uniaxialMaterial('Concrete01', matTag, fpc, epsc0, fpcu, epsU) \n
03104     Parameters (see OpenSeesPy documentation for more information): \n
03105     matTag      integer tag identifying material
03106     fpc         concrete compressive strength at 28 days (compression is negative)*
03107     epsc0       concrete strain at maximum strength*
03108     fpcu        concrete crushing strength *
03109     epsU        concrete strain at crushing strength*
03110     """
03111     uniaxialMaterial('Concrete01', ID, fc, ec, fpcu, ecu)
03112
03113
03114 def _Steel01(ID, fy, Ey, b):
03115     """
03116     Private function that generates the material model Steel01 uniaxial bilinear steel material
03117     with kinematic hardening and optional isotropic hardening described by a non-linear evolution
03118     equation.
03119     OpenSeesPy command: \n
03120     uniaxialMaterial('Steel01', matTag, fy, E0, b, a1, a2, a3, a4) \n
03121     Parameters (see OpenSeesPy documentation for more information): \n
03122     matTag      integer tag identifying material
03123     fy          yield strength
03124     E0          initial elastic tangent
03125     b           strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
03126     a1          isotropic hardening parameter, increase of compression yield envelope as proportion of yield
03127     strength after a plastic strain of a2*(fy/E0). (optional)
03128     a2          isotropic hardening parameter (see explanation under a1). (optional).
03129     a3          isotropic hardening parameter, increase of tension yield envelope as proportion of yield
03130     strength after a plastic strain of a4*(fy/E0). (optional)
03131     a4          isotropic hardening parameter (see explanation under a3). (optional)
03132     """
03133     uniaxialMaterial("Steel01", ID, fy, Ey, b)
03134
03135 def _Steel02(ID, fy, Ey, b, R0, cR1, cR2, a1, a2, a3, a4):
03136     """
03137     Private function that generates the material model Steel02 uniaxial Giuffre-Menegotto-Pinto steel
03138     material with isotropic strain hardening.
03139     OpenSeesPy command: \n
03140     uniaxialMaterial('Steel02', matTag, fy, E, b, R0, cR1, cR2, a1, a2, a3, a4, sigInit) \n
03141     Parameters (see OpenSeesPy documentation for more information): \n
03142     matTag      Integer tag identifying material
03143     fy          Yield strength
03144     E0          Initial elastic tangent
03145     b           Strain-hardening ratio (ratio between post-yield tangent and initial elastic tangent)
03146     R0 cR1 cR2  Parameters to control the transition from elastic to plastic branches.
03147     a1          Isotropic hardening parameter, increase of compression yield envelope as proportion of
03148     yield strength after a plastic strain of a2*(fy/E0). (optional)
03149     a2          Isotropic hardening parameter (see explanation under a1). (optional default = 1.0).
03150     a3          Isotropic hardening parameter, increase of tension yield envelope as proportion of
03151     yield strength after a plastic strain of a4*(fy/E0). (optional default = 0.0)
03152     a4          Isotropic hardening parameter (see explanation under a3). (optional default = 1.0)
03153     sigInit     Initial Stress Value (optional, default: 0.0) the strain is calculated from
03154     epsP=sigInit/E
03155     if (sigInit!= 0.0) { double epsInit = sigInit/E; eps = trialStrain+epsInit; } else
03156     eps = trialStrain;
03157     """
03158     uniaxialMaterial('Steel02', ID, fy, Ey, b, R0, cR1, cR2, a1, a2, a3, a4)
03159

```

```

03153
03154 def _UVCuniaxial(ID, Ey, fy, QInf, b, DInf, a, N, cK, gammaK):
03155     """
03156     Private function that generates the material model Updated Voce-Chaboche (UVC) material for
uniaxial stress states.
03157     This material is a refined version of the classic nonlinear isotropic/kinematic hardening material
model based on the Voce
03158     isotropic hardening law and the Chaboche kinematic hardening law.
03159     The UVC model contains an updated isotropic hardening law, with parameter constraints, to simulate
the permanent decrease
03160     in yield stress with initial plastic loading associated with the discontinuous yielding
phenomenon in mild steels.
03161     OpenSeesPy command: \n
03162     uniaxialMaterial('UVCuniaxial', matTag, E, fy, QInf, b, DInf, a, N, C1, gamma1, <C2 gamma2 C3
gamma3 ... C8 gamma8>) \n
03163     Parameters (see OpenSeesPy documentation for more information): \n
03164     matTag Integer tag identifying the material.
03165     E Elastic modulus of the steel material.
03166     fy Initial yield stress of the steel material.
03167     QInf Maximum increase in yield stress due to cyclic hardening (isotropic hardening).
03168     b Saturation rate of QInf, b > 0.
03169     DInf Decrease in the initial yield stress, to neglect the model updates set DInf = 0.
03170     a Saturation rate of DInf, a > 0. If DInf == 0, then a is arbitrary (but still a > 0).
03171     N Number of backstresses to define, N >= 1.
03172     cK Kinematic hardening parameter associated with backstress component k (vector).
03173     gammaK Saturation rate of kinematic hardening associated with backstress component k (vector).
03174     """
03175     backstresses = []
03176     for ii in range(N):
03177         backstresses.append(cK[ii])
03178         backstresses.append(gammaK[ii])
03179     uniaxialMaterial('UVCuniaxial', ID, Ey, fy, QInf, b, DInf, a, N, *backstresses)
03180

```

## 8.19 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/MemberModel.py File Reference ↩

### Classes

- class [ElasticElement](#)  
*Class that handles the storage and manipulation of a elastic element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.*
- class [ElasticElementSteelShape](#)  
*Class that is the children of [ElasticElement](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.*
- class [ForceBasedElement](#)  
*Class that handles the storage and manipulation of a force-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.*
- class [ForceBasedElementFibersCircRCCircShape](#)  
*Class that is the children of [ForceBasedElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.*
- class [ForceBasedElementFibersIShapeSteelShape](#)  
*Class that is the children of [ForceBasedElement](#) and combine the class [FibersIShapeSteelShape](#) (fiber section) to retrieve the information needed.*
- class [ForceBasedElementFibersRectRCRectShape](#)  
*Class that is the children of [ForceBasedElement](#) and combine the class [FibersRectRCRectShape](#) (fiber section) to retrieve the information needed.*
- class [GIFBElement](#)  
*Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.*
- class [GIFBElementFibersCircRCCircShape](#)  
*Class that is the children of [GIFBElement](#) and combine the class [FibersCircRCCircShape](#) (fiber section) to retrieve the information needed.*
- class [GIFBElementFibersRectRCRectShape](#)

Class that is the children of [GIFBElement](#) and combine the class [FibersRectRCRectShape](#) (fiber section) to retrieve the information needed.

- class [GIFBElementRCCircShape](#)

Class that is the children of [GIFBElement](#) and combine the class [RCCircShape](#) (section) to retrieve the information needed.

- class [GIFBElementRCRectShape](#)

Class that is the children of [GIFBElement](#) and combine the class [RCRectShape](#) (section) to retrieve the information needed.

- class [MemberModel](#)

Parent abstract class for the storage and manipulation of a member's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class [PanelZone](#)

Class that handles the storage and manipulation of a panel zone's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class [PanelZoneRCS](#)

WIP: Class that is the children of [PanelZone](#) and it's used for the panel zone in a RCS (RC column continous, Steel beam).

- class [PanelZoneSteelShape](#)

Class that is the children of [PanelZone](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

- class [PanelZoneSteelShapeGupta1999](#)

Class that is the children of [PanelZoneSteelShape](#) and automatically create the spring material model Gupta 1999 (ID = master\_node\_ID).

- class [PanelZoneSteelShapeSkiadopoulos2021](#)

Class that is the children of [PanelZoneSteelShape](#) and automatically create the spring material model Skiadopoulos 2021 (ID = master\_node\_ID).

- class [SpringBasedElement](#)

Class that handles the storage and manipulation of a spring-based element's information (mechanical and geometrical parameters, etc) and the initialisation in the model.

- class [SpringBasedElementModifiedIMKSteelShape](#)

Class that is the children of [SpringBasedElement](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

- class [SpringBasedElementSteelShape](#)

Class that is the children of [SpringBasedElement](#) and combine the class [SteelShape](#) (section) to retrieve the information needed.

## Namespaces

- namespace [MemberModel](#)

Module for the member model.

## Functions

- def [DefinePanelZoneElements](#) (MasterNodeID, E, RigidA, RigidI, TransfID)

Function that defines the 8 panel zone elements.

- def [DefinePanelZoneNodes](#) (int MasterNodeID, MidPanelZoneWidth, MidPanelZoneHeight)

Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master node (top center one).

## 8.20 MemberModel.py

[Go to the documentation of this file.](#)

```

00001 """
00002 Module for the member model.
00003 Carmine Schipani, 2021
00004 """
00005
00006 from openseespy.opensees import *
00007 import matplotlib.pyplot as plt
00008 import numpy as np
00009 import os
00010 import math
00011 from abc import abstractmethod
00012 from copy import copy, deepcopy
00013 from OpenSeesPyAssistant.Section import *
00014 from OpenSeesPyAssistant.DataManagement import *
00015 from OpenSeesPyAssistant.ErrorHandling import *
00016 from OpenSeesPyAssistant.Units import *
00017 from OpenSeesPyAssistant.Constants import *
00018 from OpenSeesPyAssistant.Fibers import *
00019 from OpenSeesPyAssistant.Connections import *
00020 from OpenSeesPyAssistant.FunctionalFeatures import *
00021
00022 class MemberModel(DataManagement):
00023     """
00024     Parent abstract class for the storage and manipulation of a member's information (mechanical and
00025     geometrical parameters, etc) and the initialisation in the model.
00026
00027     @param DataManagement: Parent abstract class.
00028     """
00029     @abstractmethod
00030     def Record(self, ele_ID, name_txt: str, data_dir: str, force_rec = True, def_rec = True, time_rec
00031               = True):
00032         """
00033         Abstract method that records the forces, deformation and time of the member associated with
00034         the class.
00035
00036         @param ele_ID (int): The ID of the element that will be recorded.
00037         @param name_txt (str): Name of the recorded data (no .txt).
00038         @param data_dir (str): Directory for the storage of data.
00039         @param force_rec (bool, optional): Option to record the forces (Fx, Fy, Mz). Defaults to True.
00040         @param def_rec (bool, optional): Option to record the deformation (theta) for ZeroLength
00041         element. Defaults to True.
00042         @param time_rec (bool, optional): Option to record time. Defaults to True.
00043         """
00044         if self.Initialized:
00045             if not os.path.exists(data_dir):
00046                 print("Folder {} not found in this directory; creating one".format(data_dir))
00047                 os.makedirs(data_dir)
00048
00049             if time_rec:
00050                 if force_rec:
00051                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time",
00052                             "-ele", ele_ID, "force")
00053                 if def_rec:
00054                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time",
00055                             "-ele", ele_ID, "deformation")
00056             else:
00057                 if force_rec:
00058                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele",
00059                             ele_ID, "force")
00060                 if def_rec:
00061                     recorder("Element", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-ele",
00062                             ele_ID, "deformation")
00063             else:
00064                 print("The element is not initialized (node and/or elements not created), ID =
00065                 {}".format(ele_ID))
00066
00067     @abstractmethod
00068     def RecordNodeDef(self, iNode_ID: int, jNode_ID: int, name_txt: str, data_dir: str, time_rec =
00069                       True):
00070         """
00071         Abstract method that records the deformation and time of the member's nodes associated with
00072         the class.
00073
00074         @param iNode_ID (int): ID of the node i.
00075         @param jNode_ID (int): ID of the node j.
00076         @param name_txt (str): Name of the recorded data (no .txt).
00077         @param data_dir (str): Directory for the storage of data.
00078         @param time_rec (bool, optional): Option to record time. Defaults to True.
00079         """
00080         if self.Initialized:
00081             if not os.path.exists(data_dir):
00082                 print("Folder {} not found in this directory; creating one".format(data_dir))

```



```

00072         os.makedirs(data_dir)
00073
00074         if time_rec:
00075             recorder("Node", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-time", "-node",
iNode_ID, jNode_ID, "-dof", 1, 2, 3, "disp")
00076         else:
00077             recorder("Node", "-file", '{}/{}.txt'.format(data_dir, name_txt), "-node", iNode_ID,
jNode_ID, "-dof", 1, 2, 3, "disp")
00078         else:
00079             print("The element is not initialized (node and/or elements not created), iNode ID =
{}, jNode ID = {}".format(iNode_ID, jNode_ID))
00080
00081     def _CheckL(self):
00082         """
00083         Private abstract method to check if the length of the line member is the same (with 1 cm of
00084         tolerance) with the length defined in the section used.
00085         """
00086         iNode = np.array(nodeCoord(self.iNode_ID))
00087         jNode = np.array(nodeCoord(self.jNode_ID))
00088         L = np.linalg.norm(iNode-jNode)
00089         if abs(L-self.section.L) > 1*cm_unit:
00090             print("!!!!!! WARNING !!!!!!! The length declared in the section name '{}' (L={} m) is
different from the length of the element associated (ID={}, L={}m)".format(
self.section.name_tag, L/m_unit, self.element_ID, self.section.L/m_unit))
00091
00092
00093
00094 class PanelZone(MemberModel):
00095     """
00096     Class that handles the storage and manipulation of a panel zone's information (mechanical and
geometrical parameters, etc) and the initialisation in the model.
00097
00098     @param MemberModel: Parent abstract class.
00099     """
00100     def __init__(self, master_node_ID: int, mid_panel_zone_width, mid_panel_zone_height, E, A_rigid,
I_rigid, geo_transf_ID: int, mat_ID: int, pin_corners = True):
00101         """
00102         Constructor of the class.
00103
00104         @param master_node_ID (int): ID of the master node (central top node that should be a grid
node).
00105         @param mid_panel_zone_width (float): Mid panel zone width.
00106         @param mid_panel_zone_height (float): Mid panel zone height.
00107         @param E (float): Young modulus.
00108         @param A_rigid (float): A very rigid area.
00109         @param I_rigid (float): A very rigid moment of inertia.
00110         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00111         @param mat_ID (int): ID of the material model for the panel zone spring.
00112         @param pin_corners (bool, optional): Option to pin the corners (xy03/xy04, xy06/xy07,
xy09/xy10) or not. Used for RCS models. Defaults to True.
00113
00114         @exception NegativeValue: ID needs to be a positive integer.
00115         @exception NegativeValue: mid_panel_zone_width needs to be positive.
00116         @exception NegativeValue: mid_panel_zone_height needs to be positive.
00117         @exception NegativeValue: E needs to be positive.
00118         @exception NegativeValue: A_rigid needs to be positive.
00119         @exception NegativeValue: I_rigid needs to be positive.
00120         @exception NegativeValue: geo_transf_ID needs to be a positive integer.
00121         @exception NegativeValue: mat_ID needs to be a positive integer.
00122         """
00123         # Check
00124         if master_node_ID < 1: raise NegativeValue()
00125         # if master_node_ID > 99: raise WrongNodeIDConvention(master_node_ID)
00126         if mid_panel_zone_width < 0: raise NegativeValue()
00127         if mid_panel_zone_height < 0: raise NegativeValue()
00128         if E < 0: raise NegativeValue()
00129         if A_rigid < 0: raise NegativeValue()
00130         if I_rigid < 0: raise NegativeValue()
00131         if geo_transf_ID > 1: raise NegativeValue()
00132         if mat_ID < 0: raise NegativeValue()
00133
00134         # Arguments
00135         self.master_node_IDmaster_node_ID = master_node_ID
00136         self.mid_panel_zone_widthmid_panel_zone_width = mid_panel_zone_width
00137         self.mid_panel_zone_heightmid_panel_zone_height = mid_panel_zone_height
00138         self.EE = E
00139         self.A_rigidA_rigid = A_rigid
00140         self.I_rigidI_rigid = I_rigid
00141         self.geo_transf_IDgeo_transf_ID = geo_transf_ID
00142         self.mat_IDmat_ID = mat_ID
00143         self.pin_cornerspin_corners = pin_corners
00144
00145         # Initialized the parameters that are dependent from others
00146         self.col_section_name_tagcol_section_name_tag = "None"
00147         self.beam_section_name_tagbeam_section_name_tag = "None"
00148         self.InitializedInitialized = False

```



```

00149         self.ReInitReInit()
00150
00151
00152     def ReInit(self):
00153         """
00154         Implementation of the homonym abstract method.
00155         See parent class DataManagement for detailed information.
00156         """
00157         # Arguments
00158         self.spring_IDspring_ID = -1
00159
00160         # Members
00161         if self.col_section_name_tagcol_section_name_tag != "None":
00162             self.col_section_name_tagcol_section_name_tag = self.col_section_name_tagcol_section_name_tag + "
(modified)"
00163
00164         if self.beam_section_name_tagbeam_section_name_tag != "None":
00165             self.beam_section_name_tagbeam_section_name_tag = self.beam_section_name_tagbeam_section_name_tag + "
(modified)"
00166
00167         # Data storage for loading/saving
00168         self.UpdateStoredDataUpdateStoredData()
00169
00170     # Methods
00171     def UpdateStoredData(self):
00172         """
00173         Implementation of the homonym abstract method.
00174         See parent class DataManagement for detailed information.
00175         """
00176         self.data = [{"INFO_TYPE": "PanelZone", # Tag for differentiating different data
00177             ["master_node_ID", self.master_node_IDmaster_node_ID],
00178             ["col_section_name_tag", self.col_section_name_tagcol_section_name_tag],
00179             ["beam_section_name_tag", self.beam_section_name_tagbeam_section_name_tag],
00180             ["mat_ID", self.mat_IDmat_ID],
00181             ["spring_ID", self.spring_IDspring_ID],
00182             ["mid_panel_zone_width", self.mid_panel_zone_widthmid_panel_zone_width],
00183             ["mid_panel_zone_height", self.mid_panel_zone_heightmid_panel_zone_height],
00184             ["E", self.EE],
00185             ["A_rigid", self.A_rigidA_rigid],
00186             ["I_rigid", self.I_rigidI_rigid],
00187             ["transf_ID", self.geo_transf_IDgeo_transf_ID],
00188             ["Initialized", self.InitializedInitialized]]
00189
00190     def ShowInfo(self, plot = False, block = False):
00191         """
00192         Implementation of the homonym abstract method.
00193         See parent class DataManagement for detailed information.
00194
00195         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00196
00197         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
00198         program everytime that a plot should pop up). Defaults to False.
00199         """
00200         print("")
00201         print("Requested info for Panel Zone member model, master node ID =
{}".format(self.master_node_IDmaster_node_ID))
00202         print("Section associated, column: {}".format(self.col_section_name_tagcol_section_name_tag))
00203         print("Section associated, beam: {}".format(self.beam_section_name_tagbeam_section_name_tag))
00204         print("Material model of the panel zone ID = {}".format(self.mat_IDmat_ID))
00205         print("Spring ID = {} (if -1, not defined yet)".format(self.spring_IDspring_ID))
00206         print("Mid panel zone width = {}
mm".format(self.mid_panel_zone_widthmid_panel_zone_width/mm_unit))
00207         print("Mid panel zone height = {}
mm".format(self.mid_panel_zone_heightmid_panel_zone_height/mm_unit))
00208         print("Young modulus E = {} GPa".format(self.EE/GPa_unit))
00209         print("Area of the elements (rigid) = {} mm2".format(self.A_rigidA_rigid/mm2_unit))
00210         print("Moment of inertia of the elements (strong axis, rigid) = {}
mm4".format(self.I_rigidI_rigid/mm4_unit))
00211         print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
00212         print("")
00213         if plot:
00214             if self.InitializedInitialized:
00215                 plot_member(self.element_arrayelement_array, "Panel zone, ID =
{}".format(self.master_node_IDmaster_node_ID))
00216                 if block:
00217                     plt.show()
00218             else:
00219                 print("The panel zone is not initialized (node and elements not created) for master
node ID = {}".format(self.master_node_IDmaster_node_ID))
00220
00221     def CreateMember(self):
00222         """
00223         Method that initialises the member by calling the OpenSeesPy commands through various
functions.

```

```

00223         """
00224         # Define nodes
00225         DefinePanelZoneNodes(self.master_node_IDmaster_node_ID,
self.mid_panel_zone_widthmid_panel_zone_width, self.mid_panel_zone_heightmid_panel_zone_height)
00226         xy1 = IDConvention(self.master_node_IDmaster_node_ID, 1)
00227         xy01 = IDConvention(self.master_node_IDmaster_node_ID, 1, 1)
00228         xy03 = IDConvention(self.master_node_IDmaster_node_ID, 3, 1)
00229         xy04 = IDConvention(self.master_node_IDmaster_node_ID, 4, 1)
00230         xy06 = IDConvention(self.master_node_IDmaster_node_ID, 6, 1)
00231         xy07 = IDConvention(self.master_node_IDmaster_node_ID, 7, 1)
00232         xy09 = IDConvention(self.master_node_IDmaster_node_ID, 9, 1)
00233         xy10 = IDConvention(self.master_node_IDmaster_node_ID, 10)
00234
00235         # Define rigid elements
00236         self.element_arrayelement_array = DefinePanelZoneElements(self.master_node_IDmaster_node_ID,
self.EE, self.A_rigidA_rigid, self.I_rigidI_rigid, self.geo_transf_IDgeo_transf_ID)
00237
00238         # Define zero length element
00239         self.spring_IDspring_ID = IDConvention(xy1, xy01)
00240         RotationalSpring(self.spring_IDspring_ID, xy1, xy01, self.mat_IDmat_ID)
00241         self.element_arrayelement_array.append([self.spring_IDspring_ID, xy1, xy01])
00242         self.iNode_IDiNode_ID = xy1
00243         self.jNode_IDjNode_ID = xy01
00244
00245         # Pin connections
00246         if self.pin_cornerspin_corners:
00247             Pin(xy03, xy04)
00248             Pin(xy06, xy07)
00249             Pin(xy09, xy10)
00250
00251         # Update class
00252         self.InitializedInitialized = True
00253         self.UpdateStoredDataUpdateStoredData()
00254
00255         def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00256             """
00257             Implementation of the homonym abstract method.
00258             See parent class MemberModel for detailed information.
00259             """
00260             super().Record(self.spring_IDspring_ID, name_txt, data_dir, force_rec=force_rec,
def_rec=def_rec, time_rec=time_rec)
00261
00262         def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00263             """
00264             Implementation of the homonym abstract method.
00265             See parent class MemberModel for detailed information.
00266             """
00267             super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir,
time_rec=time_rec)
00268
00269         def _CheckL(self):
00270             """
00271             (placeholder). No applicable for the panel zone.
00272             """
00273             print("No length check for panel zone")
00274
00275         class PanelZoneSteelIShape(PanelZone):
00276             """
00277             Class that is the children of PanelZone and combine the class SteelIShape (section) to retrieve
the information needed.
00278
00279             @param PanelZone: Parent class.
00280             """
00281             def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
mat_ID: int, rigid = RIGID):
00282                 """
00283                 Constructor of the class.
00284
00285                 @param master_node_ID (int): ID of the master node (central top node that should be a grid
node).
00286                 @param col (SteelIShape): SteelIShape column section object.
00287                 @param beam (SteelIShape): SteelIShape beam section object.
00288                 @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00289                 @param mat_ID (int): ID of the material model for the panel zone spring.
00290                 @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
element
but enough small to avoid convergence problem. Defaults to RIGID.
00291                 """
00292                 self.colcol = deepcopy(col)
00293                 self.beambeam = deepcopy(beam)
00294                 super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, col.E, max(col.A, beam.A)*rigid,
max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID)

```

```

00300
00301     self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
00302     self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
00303     self.UpdateStoredDataUpdateStoredData()
00304
00305
00306 class PanelZoneRCS(PanelZone):
00307     """
00308     WIP: Class that is the children of PanelZone and it's used for the panel zone in a RCS (RC column
00309     continous, Steel beam).
00310     Note that the corners are not pinned (do it manually).
00311
00312     @param PanelZone: Parent class.
00313     """
00314     def __init__(self, master_node_ID: int, col: RRectShape, beam: SteelIShape, geo_transf_ID: int,
00315     mat_ID: int, rigid = RIGID):
00316         """
00317         Constructor of the class.
00318
00319         @param master_node_ID (int): ID of the master node (central top node that should be a grid
00320         node).
00321         @param col (RRectShape): RRectShape column section object.
00322         @param beam (SteelIShape): SteelIShape beam section object.
00323         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00324         documentation).
00325         @param mat_ID (int): ID of the material model for the panel zone spring.
00326         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
00327         element
00328         but enough small to avoid convergence problem. Defaults to RIGID.
00329         """
00330         self.colcol = deepcopy(col)
00331         self.beambeam = deepcopy(beam)
00332         super().__init__(master_node_ID, col.d/2.0, beam.d/2.0, beam.E, max(col.A, beam.A)*rigid,
00333         max(col.Iy, beam.Iy)*rigid, geo_transf_ID, mat_ID, False)
00334
00335         self.col_section_name_tagcol_section_name_tagcol_section_name_tag = col.name_tag
00336         self.beam_section_name_tagbeam_section_name_tagbeam_section_name_tag = beam.name_tag
00337         self.UpdateStoredDataUpdateStoredData()
00338
00339
00340 class PanelZoneSteelIShapeGupta1999(PanelZoneSteelIShape):
00341     """
00342     Class that is the children of PanelZoneSteelIShape and automatically create the spring material
00343     model Gupta 1999 (ID = master_node_ID).
00344
00345     @param PanelZoneSteelIShape: Parent class.
00346     """
00347     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
00348     t_dp = 0, rigid=RIGID):
00349         """
00350         Constructor of the class.
00351
00352         @param master_node_ID (int): ID of the master node (central top node that should be a grid
00353         node).
00354         @param col (SteelIShape): SteelIShape column section object.
00355         @param beam (SteelIShape): SteelIShape beam section object.
00356         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
00357         documentation).
00358         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
00359         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
00360         element
00361         but enough small to avoid convergence problem. Defaults to RIGID.
00362         """
00363         self.colcolcol = deepcopy(col)
00364         self.beambeambeam = deepcopy(beam)
00365         mat_ID = master_node_ID
00366         pz_spring = Gupta1999SteelIShape(mat_ID, col, beam, t_dp)
00367         pz_spring.Hysteretic()
00368         super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)
00369
00370
00371 class PanelZoneSteelIShapeSkiadopoulos2021(PanelZoneSteelIShape):
00372     """
00373     Class that is the children of PanelZoneSteelIShape and automatically create the spring material
00374     model Skiadopoulos 2021 (ID = master_node_ID).
00375
00376     @param PanelZoneSteelIShape: Parent class.
00377     """
00378     def __init__(self, master_node_ID: int, col: SteelIShape, beam: SteelIShape, geo_transf_ID: int,
00379     t_dp = 0, rigid=RIGID):
00380         """
00381         Constructor of the class.
00382
00383         @param master_node_ID (int): ID of the master node (central top node that should be a grid
00384         node).
00385         @param col (SteelIShape): SteelIShape column section object.

```

```

00373         @param beam (SteelIShape): SteelIShape beam section object.
00374         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00375         @param t_dp (float, optional): Doubler plate thickness. Defaults to 0.
00376         @param rigid (float, optional): Parameter with a value enough big to assure rigidity of one
element
00377         but enough small to avoid convergence problem. Defaults to RIGID.
00378         """
00379         self.colcolcol = deepcopy(col)
00380         self.beambeambeam = deepcopy(beam)
00381         mat_ID = master_node_ID
00382         pz_spring = Skiadopoulos2021SteelIShape(mat_ID, col, beam, t_dp)
00383         pz_spring.Hysteretic()
00384
00385         super().__init__(master_node_ID, col, beam, geo_transf_ID, mat_ID, rigid)
00386
00387
00388 def DefinePanelZoneNodes(MasterNodeID: int, MidPanelZoneWidth, MidPanelZoneHeight):
00389     """
00390     Function that defines the remaining 10 nodes of a panel zone given the dimensions and the master
node (top center one).
00391     ID convention for the panel zone: \n
00392         PZNodeID:      12 nodes: top right lxy (master), lxy1 top right,
lxy09,lxy10      lxy      lxy1,lxy01 \n
00393         clockwise 10 nodes xy01-xy10 (with double node at corners)
o-----o-----o      \n
00394         Spring at node lxy1
|      \n
00395         PZElementID:  8 elements: starting at node lxy, clockwise
|      \n
00396         (see function DefinePanelZoneElements for more info)
|      \n
00397         |      \n
00398         |      \n
o lxy02 \n
00399         |      \n
0400         |      \n
0401         |      \n
0402         |      \n
0403         |      \n
o-----o-----o      \n
0404         lxy06,lxy07      lxy05      lxy03,lxy04 \n
Note that the top right node is defined differently because is where the spring is.
0405
0406         @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
0407         @param MidPanelZoneWidth (float): Mid panel zone width.
0408         @param MidPanelZoneHeight (float): Mid panel zone height.
0409
0410         """
0411         # Get node coord and define useful variables
0412         m_node = np.array(nodeCoord(MasterNodeID))
0413         AxisCL = m_node[0]
0414         FloorCL = m_node[1] - MidPanelZoneHeight
0415
0416         # Convention: Node of the spring (top right) is xyl
0417         node(IDConvention(MasterNodeID, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
0418         # Convention: Two nodes in the corners (already defined one, xyl) clockwise from xy01 to xy10
0419         node(IDConvention(MasterNodeID, 1, 1), AxisCL+MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
0420         node(IDConvention(MasterNodeID, 2, 1), AxisCL+MidPanelZoneWidth, FloorCL)
0421         node(IDConvention(MasterNodeID, 3, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
0422         node(IDConvention(MasterNodeID, 4, 1), AxisCL+MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
0423         node(IDConvention(MasterNodeID, 5, 1), AxisCL, FloorCL-MidPanelZoneHeight)
0424         node(IDConvention(MasterNodeID, 6, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
0425         node(IDConvention(MasterNodeID, 7, 1), AxisCL-MidPanelZoneWidth, FloorCL-MidPanelZoneHeight)
0426         node(IDConvention(MasterNodeID, 8, 1), AxisCL-MidPanelZoneWidth, FloorCL)
0427         node(IDConvention(MasterNodeID, 9, 1), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
0428         node(IDConvention(MasterNodeID, 10), AxisCL-MidPanelZoneWidth, FloorCL+MidPanelZoneHeight)
0429
0430
0431
0432 def DefinePanelZoneElements(MasterNodeID, E, RigidA, RigidI, TransfID):
0433     """
0434     Function that defines the 8 panel zone elements. For the ID convention, see DefinePanelZoneNodes.
0435
0436     @param MasterNodeID (int): ID of the master node (central top node that should be a grid node).
0437     @param E (float): Young modulus.
0438     @param RigidA (float): A very rigid area.
0439     @param RigidI (float): A very rigid moment of inertia.
0440     @param TransfID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
0441
0442     @returns list: List of lists, with each list containing the ID of the element, of node i and node

```

```

j.
00443     """
00444     # Compute the ID of the nodes obeying to the convention used
00445     xy = MasterNodeID
00446     xy1 = IDConvention(xy, 1)
00447     xy01 = IDConvention(xy, 1, 1)
00448     xy02 = IDConvention(xy, 2, 1)
00449     xy03 = IDConvention(xy, 3, 1)
00450     xy04 = IDConvention(xy, 4, 1)
00451     xy05 = IDConvention(xy, 5, 1)
00452     xy06 = IDConvention(xy, 6, 1)
00453     xy07 = IDConvention(xy, 7, 1)
00454     xy08 = IDConvention(xy, 8, 1)
00455     xy09 = IDConvention(xy, 9, 1)
00456     xy10 = IDConvention(xy, 10)
00457
00458     # Create element IDs using the convention: xy(a)xy(a) with xy(a) = NodeID i and j
00459     # Starting at MasterNodeID, clockwise
00460     # if MasterNodeID > 99:
00461     #     print("Warning, convention: MasterNodeID's digits should be 2")
00462
00463     ele1 = IDConvention(xy, xy1)
00464     ele2 = IDConvention(xy01, xy02)
00465     ele3 = IDConvention(xy02, xy03)
00466     ele4 = IDConvention(xy04, xy05)
00467     ele5 = IDConvention(xy05, xy06)
00468     ele6 = IDConvention(xy07, xy08)
00469     ele7 = IDConvention(xy08, xy09)
00470     ele8 = IDConvention(xy10, xy)
00471
00472     # Create panel zone elements
00473     #
00474     element("elasticBeamColumn", ele1, xy, xy1, RigidA, E, RigidI, TransfID)
00475     element("elasticBeamColumn", ele2, xy01, xy02, RigidA, E, RigidI, TransfID)
00476     element("elasticBeamColumn", ele3, xy02, xy03, RigidA, E, RigidI, TransfID)
00477     element("elasticBeamColumn", ele4, xy04, xy05, RigidA, E, RigidI, TransfID)
00478     element("elasticBeamColumn", ele5, xy05, xy06, RigidA, E, RigidI, TransfID)
00479     element("elasticBeamColumn", ele6, xy07, xy08, RigidA, E, RigidI, TransfID)
00480     element("elasticBeamColumn", ele7, xy08, xy09, RigidA, E, RigidI, TransfID)
00481     element("elasticBeamColumn", ele8, xy10, xy, RigidA, E, RigidI, TransfID)
00482
00483     # Create element array for further manipulations
00484     element_array = [[ele1, xy, xy1],
00485                     [ele2, xy01, xy02],
00486                     [ele3, xy02, xy03],
00487                     [ele4, xy04, xy05],
00488                     [ele5, xy05, xy06],
00489                     [ele6, xy07, xy08],
00490                     [ele7, xy08, xy09],
00491                     [ele8, xy10, xy]]
00492
00493     return element_array
00494
00495
00496 class ElasticElement(MemberModel):
00497     """
00498     Class that handles the storage and manipulation of a elastic element's information (mechanical and
    geometrical parameters, etc) and the initialisation in the model.
00499
00500     @param MemberModel: Parent abstract class.
00501     """
00502     def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy, geo_transf_ID: int, ele_ID = -1):
00503         """
00504         Constructor of the class.
00505
00506         @param iNode_ID (int): ID of the first end node.
00507         @param jNode_ID (int): ID of the second end node.
00508         @param A (float): Area of the member.
00509         @param E (float): Young modulus.
00510         @param Iy (float): Second moment of inertia (strong axis).
00511         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
    documentation).
00512         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
    IDConvention to define it.
00513
00514         @exception NegativeValue: ID needs to be a positive integer.
00515         @exception NegativeValue: ID needs to be a positive integer.
00516         @exception NegativeValue: A needs to be positive.
00517         @exception NegativeValue: E needs to be positive.
00518         @exception NegativeValue: Iy needs to be positive.
00519         @exception NegativeValue: ID needs to be a positive integer.
00520         @exception NegativeValue: ID needs to be a positive integer.
00521         """
00522         # Check
00523         if iNode_ID < 1: raise NegativeValue()
00524         if jNode_ID < 1: raise NegativeValue()
00525         if A < 0: raise NegativeValue()

```

```

00526         if E < 0: raise NegativeValue()
00527         if Iy < 0: raise NegativeValue()
00528         if geo_transf_ID < 1: raise NegativeValue()
00529         if ele_ID != -1 and ele_ID < 1: raise NegativeValue()
00530
00531         # Arguments
00532         self.iNode_IDiNode_ID = iNode_ID
00533         self.jNode_IDjNode_ID = jNode_ID
00534         self.AA = A
00535         self.EE = E
00536         self.IyIy = Iy
00537         self.geo_transf_IDgeo_transf_ID = geo_transf_ID
00538
00539         # Initialized the parameters that are dependent from others
00540         self.section_name_tagsection_name_tag = "None"
00541         self.InitializedInitialized = False
00542         self.ReInitReInit(ele_ID = -1)
00543
00544     def ReInit(self, ele_ID = -1):
00545         """
00546         Implementation of the homonym abstract method.
00547         See parent class DataManagement for detailed information.
00548
00549         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
00550         IDConvention to define it.
00551         """
00552         # Members
00553         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
00554         self.section_name_tagsection_name_tag + " (modified)"
00555
00556         # element ID
00557         self.element_IDelement_ID = IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID) if
00558         ele_ID == -1 else ele_ID
00559
00560         # Data storage for loading/saving
00561         self.UpdateStoredDataUpdateStoredData()
00562
00563     # Methods
00564     def UpdateStoredData(self):
00565         """
00566         Implementation of the homonym abstract method.
00567         See parent class DataManagement for detailed information.
00568         """
00569         self.datadata = [{"INFO_TYPE", "ElasticElement"}, # Tag for differentiating different data
00570         [{"element_ID", self.element_IDelement_ID},
00571         [{"section_name_tag", self.section_name_tagsection_name_tag},
00572         [{"A", self.AA},
00573         [{"E", self.EE},
00574         [{"Iy", self.IyIy},
00575         [{"iNode_ID", self.iNode_IDiNode_ID},
00576         [{"jNode_ID", self.jNode_IDjNode_ID},
00577         [{"tranf_ID", self.geo_transf_IDgeo_transf_ID},
00578         [{"Initialized", self.InitializedInitialized}]
00579
00580     def ShowInfo(self, plot = False, block = False):
00581         """
00582         Implementation of the homonym abstract method.
00583         See parent class DataManagement for detailed information.
00584
00585         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
00586         False.
00587         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
00588         of the program everytime that a plot should pop up). Defaults to False.
00589         """
00590         print("")
00591         print("Requested info for ElasticElement member model, ID =
00592         {}".format(self.element_IDelement_ID))
00593         print("Section associated {}".format(self.section_name_tagsection_name_tag))
00594         print("Area A = {} mm2".format(self.AA/mm2_unit))
00595         print("Young modulus E = {} GPa".format(self.EE/GPa_unit))
00596         print("Moment of inertia Iy = {} mm4".format(self.IyIy/mm4_unit))
00597         print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
00598         print("")
00599         if plot:
00600             if self.InitializedInitialized:
00601                 plot_member(self.element_arrayelement_array, "Elastic Element, ID =
00602                 {}".format(self.element_IDelement_ID))
00603                 if block:
00604                     plt.show()
00605             else:
00606                 print("The ElasticElement is not initialized (node and elements not created), ID =
00607                 {}".format(self.element_IDelement_ID))
00608
00609
00610

```

```

00605     def CreateMember(self):
00606         """
00607         Method that initialises the member by calling the OpenSeesPy commands through various
functions.
00608         """
00609         self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_IDiNode_ID,
self.jNode_IDjNode_ID]]
00610
00611         # Define element
00612         element("elasticBeamColumn", self.element_IDelement_ID, self.iNode_IDiNode_ID,
self.jNode_IDjNode_ID, self.AA, self.EE, self.IyIy, self.geo_transf_IDgeo_transf_ID)
00613
00614         # Update class
00615         self.InitializedInitialized = True
00616         self.UpdateStoredDataUpdateStoredData()
00617
00618
00619     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
00620         """
00621         Implementation of the homonym abstract method.
00622         See parent class MemberModel for detailed information.
00623         """
00624         super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec,
def_rec=def_rec, time_rec=time_rec)
00625
00626
00627     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00628         """
00629         Implementation of the homonym abstract method.
00630         See parent class MemberModel for detailed information.
00631         """
00632         super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir,
time_rec=time_rec)
00633
00634
00635 class ElasticElementSteelIShape(ElasticElement):
00636     """
00637     Class that is the children of ElasticElement and combine the class SteelIShape (section) to
retrieve the information needed.
00638
00639     @param ElasticElement: Parent class.
00640     """
00641     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int, ele_ID
= -1):
00642         """
00643         Constructor of the class.
00644
00645         @param iNode_ID (int): ID of the first end node.
00646         @param jNode_ID (int): ID of the second end node.
00647         @param section (SteelIShape): SteelIShape section object.
00648         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00649         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00650         """
00651         self.sectionsection = deepcopy(section)
00652         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy, geo_transf_ID, ele_ID)
00653         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
00654         self.UpdateStoredDataUpdateStoredData()
00655         # Check length
00656         self._CheckL_CheckL()
00657
00658
00659 class SpringBasedElement(MemberModel):
00660     """
00661     Class that handles the storage and manipulation of a spring-based element's information
(mechanical and geometrical parameters, etc) and the initialisation in the model.
00662
00663     @param MemberModel: Parent abstract class.
00664     """
00665     def __init__(self, iNode_ID: int, jNode_ID: int, A, E, Iy_mod, geo_transf_ID: int, mat_ID_i = -1,
mat_ID_j = -1, ele_ID = -1):
00666         """
00667         Constructor of the class.
00668
00669         @param iNode_ID (int): ID of the first end node.
00670         @param jNode_ID (int): ID of the second end node.
00671         @param A (float): Area of the member.
00672         @param E (float): Young modulus.
00673         @param Iy_mod (float): Second moment of inertia (strong axis).
00674         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00675         @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if
present). Defaults to -1.
00676         @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if
present). Defaults to -1.
00677         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use

```

```

IDConvention to define it.
00678
00679     @exception NegativeValue: ID needs to be a positive integer.
00680     @exception NegativeValue: ID needs to be a positive integer.
00681     @exception NegativeValue: A needs to be positive.
00682     @exception NegativeValue: E needs to be positive.
00683     @exception NegativeValue: Iy_mod needs to be positive.
00684     @exception NegativeValue: ID needs to be a positive integer.
00685     @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00686     @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00687     @exception NameError: at least one spring needs to be defined.
00688     @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00689     """
00690     # Check
00691     if iNode_ID < 1: raise NegativeValue()
00692     if jNode_ID < 1: raise NegativeValue()
00693     if A < 0: raise NegativeValue()
00694     if E < 0: raise NegativeValue()
00695     if Iy_mod < 0: raise NegativeValue()
00696     if geo_transf_ID < 1: raise NegativeValue()
00697     if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
00698     if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
00699     if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID =
{}").format(IDConvention(iNode_ID, jNode_ID))
00700     if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00701
00702     # Arguments
00703     self.iNode_IDiNode_ID = iNode_ID
00704     self.jNode_IDjNode_ID = jNode_ID
00705     self.AA = A
00706     self.EE = E
00707     self.Iy_modIy_mod = Iy_mod
00708     self.geo_transf_IDgeo_transf_ID = geo_transf_ID
00709     self.mat_IDimat_ID_i = mat_ID_i
00710     self.mat_IDjmat_ID_j = mat_ID_j
00711
00712     # Initialized the parameters that are dependent from others
00713     self.section_name_tagsection_name_tag = "None"
00714     self.InitializedInitialized = False
00715     self.ReInitReInit(ele_ID)
00716
00717
00718     def ReInit(self, ele_ID = -1):
00719         """
00720         Implementation of the homonym abstract method.
00721         See parent class DataManagement for detailed information.
00722
00723         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00724         """
00725         # Members
00726         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
00727         # orientation:
00728         self.ele_orientationele_orientation = NodesOrientation(self.iNode_IDiNode_ID,
self.jNode_IDjNode_ID)
00729         if self.ele_orientationele_orientation == "zero_length": raise
ZeroLength(IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID))
00730
00731         if self.mat_IDimat_ID_i != -1:
00732             self.iNode_ID_springiNode_ID_spring = OffsetNodeIDConvention(self.iNode_IDiNode_ID,
self.ele_orientationele_orientation, "i")
00733         else:
00734             self.iNode_ID_springiNode_ID_spring = self.iNode_IDiNode_ID
00735
00736         if self.mat_IDjmat_ID_j != -1:
00737             self.jNode_ID_springjNode_ID_spring = OffsetNodeIDConvention(self.jNode_IDjNode_ID,
self.ele_orientationele_orientation, "j")
00738         else:
00739             self.jNode_ID_springjNode_ID_spring = self.jNode_IDjNode_ID
00740
00741         # element ID
00742         self.element_IDelement_ID = IDConvention(self.iNode_ID_springiNode_ID_spring,
self.jNode_ID_springjNode_ID_spring) if ele_ID == -1 else ele_ID
00743
00744         # Data storage for loading/saving
00745         self.UpdateStoredDataUpdateStoredData()
00746
00747
00748     # Methods
00749     def UpdateStoredData(self):
00750         """
00751         Implementation of the homonym abstract method.
00752         See parent class DataManagement for detailed information.
00753         """
00754         self.datadata = [{"INFO_TYPE", "SpringBasedElement"}, # Tag for differentiating different data
00755         ["element_ID", self.element_IDelement_ID],

```



```

00756         ["section_name_tag", self.section_name_tagsection_name_tag],
00757         ["A", self.AA],
00758         ["E", self.EE],
00759         ["Iy_mod", self.Iy_modIy_mod],
00760         ["iNode_ID", self.iNode_IDiNode_ID],
00761         ["iNode_ID_spring", self.iNode_ID_springiNode_ID_spring],
00762         ["mat_ID_i", self.mat_ID_imat_ID_i],
00763         ["jNode_ID", self.jNode_IDjNode_ID],
00764         ["jNode_ID_spring", self.jNode_ID_springjNode_ID_spring],
00765         ["mat_ID_j", self.mat_ID_jmat_ID_j],
00766         ["ele_orientation", self.ele_orientationele_orientation],
00767         ["tridf_ID", self.geo_transf_IDgeo_transf_ID],
00768         ["Initialized", self.InitializedInitialized]]
00769
00770
00771     def ShowInfo(self, plot = False, block = False):
00772         """
00773         Implementation of the homonym abstract method.
00774         See parent class DataManagement for detailed information.
00775
00776         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
00777         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the
program everytime that a plot should pop up). Defaults to False.
00778         """
00779         print("")
00780         print("Requested info for SpringBasedElement member model, ID =
{}".format(self.element_IDelement_ID))
00781         print("Section associated {} ".format(self.section_name_tagsection_name_tag))
00782         print("Material model of the spring i, ID = {}".format(self.mat_ID_imat_ID_i))
00783         print("Material model of the spring j, ID = {}".format(self.mat_ID_jmat_ID_j))
00784         print("Area A = {} mm2".format(self.AA/mm2_unit))
00785         print("Young modulus E = {} GPa".format(self.EE/GPa_unit))
00786         print("n modified moment of inertia Iy_mod = {} mm4".format(self.Iy_modIy_mod/mm4_unit))
00787         print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
00788         print("")
00789
00790         if plot:
00791             if self.InitializedInitialized:
00792                 plot_member(self.element_arrayelement_array, "SpringBased Element, ID =
{}".format(self.element_IDelement_ID))
00793                 if block:
00794                     plt.show()
00795             else:
00796                 print("The SpringBasedElement is not initialized (node and elements not created), ID =
{}".format(self.element_IDelement_ID))
00797
00798
00799     def CreateMember(self):
00800         """
00801         Method that initialises the member by calling the OpenSeesPy commands through various
functions.
00802         """
00803         self.element_arrayelement_array = [[self.element_IDelement_ID,
self.iNode_ID_springiNode_ID_spring, self.jNode_ID_springjNode_ID_spring]]
00804         if self.mat_ID_imat_ID_i != -1:
00805             # Define zero length element i
00806             node(self.iNode_ID_springiNode_ID_spring, *nodeCoord(self.iNode_IDiNode_ID))
00807             self.iSpring_IDiSpring_ID = IDConvention(self.iNode_IDiNode_ID,
self.iNode_ID_springiNode_ID_spring)
00808             RotationalSpring(self.iSpring_IDiSpring_ID, self.iNode_IDiNode_ID,
self.iNode_ID_springiNode_ID_spring, self.mat_ID_imat_ID_i)
00809             self.element_arrayelement_array.append([self.iSpring_IDiSpring_ID, self.iNode_IDiNode_ID,
self.iNode_ID_springiNode_ID_spring])
00810             if self.mat_ID_jmat_ID_j != -1:
00811                 # Define zero length element j
00812                 node(self.jNode_ID_springjNode_ID_spring, *nodeCoord(self.jNode_IDjNode_ID))
00813                 self.jSpring_IDjSpring_ID = IDConvention(self.jNode_IDjNode_ID,
self.jNode_ID_springjNode_ID_spring)
00814                 RotationalSpring(self.jSpring_IDjSpring_ID, self.jNode_IDjNode_ID,
self.jNode_ID_springjNode_ID_spring, self.mat_ID_jmat_ID_j)
00815                 self.element_arrayelement_array.append([self.jSpring_IDjSpring_ID, self.jNode_IDjNode_ID,
self.jNode_ID_springjNode_ID_spring])
00816
00817             # Define element
00818             element("elasticBeamColumn", self.element_IDelement_ID, self.iNode_ID_springiNode_ID_spring,
self.jNode_ID_springjNode_ID_spring, self.AA, self.EE, self.Iy_modIy_mod,
self.geo_transf_IDgeo_transf_ID)
00819
00820             # Update class
00821             self.InitializedInitialized = True
00822             self.UpdateStoredDataUpdateStoredData()
00823
00824
00825     def Record(self, spring_or_element: str, name_txt: str, data_dir: str, force_rec=True,
def_rec=True, time_rec=True):
00826         """

```

```

00827         Implementation of the homonym abstract method.
00828         See parent class MemberModel for detailed information.
00829         """
00830         if spring_or_element == "element":
00831             super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec,
def_rec=def_rec, time_rec=time_rec)
00832         elif spring_or_element == "spring_i":
00833             if self.mat_ID_imat_ID_i == -1:
00834                 print("Spring i recorded not present in element ID =
{}".format(self.element_IDelement_ID))
00835             else:
00836                 super().Record(self.iSpring_IDiSpring_ID, name_txt, data_dir, force_rec=force_rec,
def_rec=def_rec, time_rec=time_rec)
00837         elif spring_or_element == "spring_j":
00838             if self.mat_ID_jmat_ID_j == -1:
00839                 print("Spring j recorded not present in element ID =
{}".format(self.element_IDelement_ID))
00840             else:
00841                 super().Record(self.jSpring_IDjSpring_ID, name_txt, data_dir, force_rec=force_rec,
def_rec=def_rec, time_rec=time_rec)
00842         else:
00843             print("No recording option with: '{}' with element ID: {}".format(spring_or_element,
self.element_IDelement_ID))
00844
00845     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
00846         """
00847         Implementation of the homonym abstract method.
00848         See parent class MemberModel for detailed information.
00849         """
00850         super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir,
time_rec=time_rec)
00851
00852
00853
00854 class SpringBasedElementSteelIShape(SpringBasedElement):
00855     """
00856     Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to
retrieve the information needed.
00857     L_b is assumed the same for top and bottom springs.
00858
00859     @param SpringBasedElement: Parent class.
00860     """
00861     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int,
mat_ID_i=-1, mat_ID_j=-1, ele_ID = -1):
00862         """
00863         Constructor of the class.
00864
00865         @param iNode_ID (int): ID of the first end node.
00866         @param jNode_ID (int): ID of the second end node.
00867         @param section (SteelIShape): SteelIShape section object.
00868         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00869         @param mat_ID_i (int, optional): ID of the material model for the spring in the node i (if
present). Defaults to -1.
00870         @param mat_ID_j (int, optional): ID of the material model for the spring in the node j (if
present). Defaults to -1.
00871         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00872
00873         @exception NegativeValue: ID needs to be a positive integer.
00874         @exception NegativeValue: ID needs to be a positive integer.
00875         @exception NameError: at least one spring needs to be defined.
00876         @exception NegativeValue: ID needs to be a positive integer.
00877         """
00878         self.sectionsection = deepcopy(section)
00879         if mat_ID_i != -1 and mat_ID_i < 0: raise NegativeValue()
00880         if mat_ID_j != -1 and mat_ID_j < 0: raise NegativeValue()
00881         if mat_ID_i == -1 and mat_ID_j == -1: raise NameError("No springs defined for element ID =
{}".format(IDConvention(iNode_ID, jNode_ID)))
00882         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00883
00884         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID,
mat_ID_i=mat_ID_i, mat_ID_j=mat_ID_j, ele_ID=ele_ID)
00885         self.section_name_tagsection_name_tag = section.name_tag
00886         self.UpdateStoredDataUpdateStoredData()
00887         # Check length
00888         self._CheckL_CheckL()
00889
00890
00891 class SpringBasedElementModifiedIMKSteelIShape(SpringBasedElement):
00892     """
00893     Class that is the children of SpringBasedElement and combine the class SteelIShape (section) to
retrieve the information needed.
00894     If there are two springs and the inflection point not in the middle, use two spring elements,
connect them rigidly in the inflection point with one spring each in the extremes.
00895     L_b is assumed the same for top and bottom springs.
00896

```

```

00897     @param SpringBasedElement: Parent class.
00898     """
00899     def __init__(self, iNode_ID: int, jNode_ID: int, section: SteelIShape, geo_transf_ID: int,
new_mat_ID_i=-1, new_mat_ID_j=-1,
00900         N_G = 0, L_0 = -1, L_b = -1, ele_ID = -1):
00901         """
00902         Constructor of the class.
00903
00904         @param iNode_ID (int): ID of the first end node.
00905         @param jNode_ID (int): ID of the second end node.
00906         @param section (SteelIShape): SteelIShape section object.
00907         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
00908         @param new_mat_ID_i (int, optional): New ID for the definition of the material model for the
spring in the node i.
00909             If -1 is passed, the class generate no material model and no spring. If 0 is passed, no i
spring. Defaults to -1.
00910         @param new_mat_ID_j (int, optional): New ID for the definition of the material model for the
spring in the node j.
00911             If -1 is passed, the class generate no material model and no spring. If 0 is passed, no j
spring. Defaults to -1.
00912         @param N_G (float, optional): Axial load. Defaults to 0.
00913         @param L_0 (float, optional): Distance from the maximal moment to zero. Defaults to -1, e.g.
computed in __init__().
00914         @param L_b (float, optional): Maximal unbraced lateral buckling length. Defaults to -1, e.g.
computed in __init__().
00915         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00916
00917         @exception NegativeValue: ID needs to be a positive integer.
00918         @exception NegativeValue: ID needs to be a positive integer.
00919         @exception NameError: at least one spring needs to be defined.
00920         @exception NegativeValue: ID needs to be a positive integer.
00921         @exception ZeroLength: The two nodes are superimposed.
00922         """
00923         self.sectionsection = deepcopy(section)
00924         if new_mat_ID_i != -1 and new_mat_ID_i < 0: raise NegativeValue()
00925         if new_mat_ID_j != -1 and new_mat_ID_j < 0: raise NegativeValue()
00926         if new_mat_ID_i == 0 and new_mat_ID_j == 0: raise NameError("No springs imposed for element ID
= {}. Use ElasticElement instead".format(IDConvention(iNode_ID, jNode_ID)))
00927         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
00928
00929         if L_0 == -1:
00930             if new_mat_ID_i != 0 and new_mat_ID_j != 0:
00931                 L_0 = section.L/2
00932             else:
00933                 L_0 = section.L
00934         L_b = L_0 if L_b == -1 else L_b
00935
00936         # auto assign ID for material of springs
00937         ele_orientation = NodesOrientation(iNode_ID, jNode_ID)
00938         if ele_orientation == "zero_length": raise ZeroLength(IDConvention(iNode_ID, jNode_ID))
00939         if new_mat_ID_i != 0 and new_mat_ID_i == -1:
00940             new_mat_ID_i = OffsetNodeIDConvention(iNode_ID, ele_orientation, "i")
00941         if new_mat_ID_j != 0 and new_mat_ID_j == -1:
00942             new_mat_ID_j = OffsetNodeIDConvention(jNode_ID, ele_orientation, "j")
00943
00944         if new_mat_ID_i != 0:
00945             # Create mat i
00946             iSpring = ModifiedIMKSteelIShape(new_mat_ID_i, section, N_G, L_0 = L_0, L_b = L_b)
00947             iSpring.Bilin()
00948
00949         if new_mat_ID_j != 0:
00950             # Create mat j
00951             jSpring = ModifiedIMKSteelIShape(new_mat_ID_j, section, N_G, L_0 = L_0, L_b = L_b)
00952             jSpring.Bilin()
00953
00954         new_mat_ID_i = -1 if new_mat_ID_i == 0 else new_mat_ID_i
00955         new_mat_ID_j = -1 if new_mat_ID_j == 0 else new_mat_ID_j
00956
00957         super().__init__(iNode_ID, jNode_ID, section.A, section.E, section.Iy_mod, geo_transf_ID,
mat_ID_i=new_mat_ID_i, mat_ID_j=new_mat_ID_j, ele_ID=ele_ID)
00958         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
00959         self.UpdateStoredDataUpdateStoredData()
00960         # Check length
00961         self._CheckL_CheckL()
00962
00963
00964 class ForceBasedElement(MemberModel):
00965     """
00966     Class that handles the storage and manipulation of a force-based element's information (mechanical
and geometrical parameters, etc) and the initialisation in the model.
00967
00968     @param MemberModel: Parent abstract class.
00969     """
00970     def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, geo_transf_ID: int,
new_integration_ID = -1, Ip = 5, integration_type = "Lobatto", max_iter =
00971

```

```

MAX_ITER_INTEGRATION, tol = TOL_INTEGRATION, ele_ID = -1):
00972     """
00973     Constructor of the class.
00974
00975     @param iNode_ID (int): ID of the first end node.
00976     @param jNode_ID (int): ID of the second end node.
00977     @param fiber_ID (int): ID of the fiber section.
00978     @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
documentation).
00979     @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
00980     @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
00981     @param integration_type (str, optional): Integration type. For more information, see
OpenSeesPy documentation.
00982         Defaults to "Lobatto".
00983     @param max_iter (int, optional): Maximal number of iteration to reach the integration
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
00984     @param tol (float, optional): Tolerance for the integration convergence. Defaults to
TOL_INTEGRATION (Units).
00985     @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
00986
00987     @exception NegativeValue: ID needs to be a positive integer.
00988     @exception NegativeValue: ID needs to be a positive integer.
00989     @exception NegativeValue: ID needs to be a positive integer.
00990     @exception NegativeValue: ID needs to be a positive integer.
00991     @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00992     @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from
-1.
00993
00994     @exception NegativeValue: max_iter needs to be a positive integer.
00995     @exception NegativeValue: tol needs to be positive.
00996     @exception NegativeValue: ID needs to be a positive integer, if different from -1.
00997     """
00998     # Check
00999     if iNode_ID < 1: raise NegativeValue()
01000     if jNode_ID < 1: raise NegativeValue()
01001     if fiber_ID < 1: raise NegativeValue()
01002     if geo_transf_ID < 1: raise NegativeValue()
01003     if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
01004     if Ip != -1 and Ip < 3: raise NegativeValue()
01005     if max_iter < 0: raise NegativeValue()
01006     if tol < 0: raise NegativeValue()
01007     if ele_ID != -1 and ele_ID < 1: raise NegativeValue()
01008
01009     # Arguments
01010     self.iNode_IDiNode_ID = iNode_ID
01011     self.jNode_IDjNode_ID = jNode_ID
01012     self.fiber_IDfiber_ID = fiber_ID
01013     self.geo_transf_IDgeo_transf_ID = geo_transf_ID
01014     self.IpIp = Ip
01015     self.integration_typeintegration_type = integration_type
01016     self.max_itermax_iter = max_iter
01017     self.toltol = tol
01018
01019     # Initialized the parameters that are dependent from others
01020     self.section_name_tagsection_name_tag = "None"
01021     self.InitializedInitialized = False
01022     self.ReInitReInit(new_integration_ID, ele_ID)
01023
01024     def ReInit(self, new_integration_ID, ele_ID = -1):
01025         """
01026         Implementation of the homonym abstract method.
01027         See parent class DataManagement for detailed information.
01028
01029         @param new_integration_ID (int): ID of the integration technique.
01030         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01031         """
01032         # Precompute some members
01033         self.element_IDelement_ID = IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID) if
ele_ID == -1 else ele_ID
01034
01035         # Arguments
01036         self.new_integration_IDnew_integration_ID = self.element_IDelement_ID if new_integration_ID ==
-1 else new_integration_ID
01037
01038         # Members
01039         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =
self.section_name_tagsection_name_tag + " (modified)"
01040
01041         # Data storage for loading/saving
01042         self.UpdateStoredDataUpdateStoredData()
01043
01044         # Methods
01045         def UpdateStoredData(self):

```

```

01047     """
01048     Implementation of the homonym abstract method.
01049     See parent class DataManagement for detailed information.
01050     """
01051     self.datadata = [{"INFO_TYPE", "ForceBasedElement"}, # Tag for differentiating different data
01052                     [{"element_ID", self.element_IDelement_ID},
01053                      [{"section_name_tag", self.section_name_tagsection_name_tag},
01054                       [{"Ip", self.IpIp},
01055                        [{"iNode_ID", self.iNode_IDiNode_ID},
01056                         [{"jNode_ID", self.jNode_IDjNode_ID},
01057                          [{"fiber_ID", self.fiber_IDfiber_ID},
01058                           [{"new_integration_ID", self.new_integration_IDnew_integration_ID},
01059                            [{"integration_type", self.integration_typeintegration_type},
01060                             [{"tol", self.toltol},
01061                              [{"max_iter", self.max_itermax_iter},
01062                               [{"tranf_ID", self.geo_transf_IDgeo_transf_ID},
01063                                [{"Initialized", self.InitializedInitialized}]]]]]]]]]]]]]]]
01064
01065
01066     def ShowInfo(self, plot = False, block = False):
01067         """
01068         Implementation of the homonym abstract method.
01069         See parent class DataManagement for detailed information.
01070
01071         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
01072         False.
01073         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
01074         of the program everytime that a plot should pop up). Defaults to False.
01075         """
01076         print("")
01077         print("Requested info for ForceBasedElement member model, ID =
01078         {}".format(self.element_IDelement_ID))
01079         print("Fiber associated, ID = {}".format(self.fiber_IDfiber_ID))
01080         print("Integration type '{}', ID = {}".format(self.integration_typeintegration_type,
01081         self.new_integration_IDnew_integration_ID))
01082         print("Section associated {} ".format(self.section_name_tagsection_name_tag))
01083         print("Number of integration points along the element Ip = {}, max iter = {}, tol =
01084         {}".format(self.IpIp, self.max_itermax_iter, self.toltol))
01085         print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
01086         print("")
01087         if plot:
01088             if self.InitializedInitialized:
01089                 plot_member(self.element_arrayelement_array, "ForceBased Element, ID =
01090                 {}".format(self.element_IDelement_ID))
01091                 if block:
01092                     plt.show()
01093             else:
01094                 print("The ForceBasedElement is not initialized (element not created), ID =
01095                 {}".format(self.element_IDelement_ID))
01096
01097
01098     def CreateMember(self):
01099         """
01100         Method that initialises the member by calling the OpenSeesPy commands through various
01101         functions.
01102         """
01103         self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_IDiNode_ID,
01104         self.jNode_IDjNode_ID]]
01105
01106         # Define integration type
01107         beamIntegration(self.integration_typeintegration_type,
01108         self.new_integration_IDnew_integration_ID, self.fiber_IDfiber_ID, self.IpIp)
01109
01110         # Define element
01111         element('forceBeamColumn', self.element_IDelement_ID, self.iNode_IDiNode_ID,
01112         self.jNode_IDjNode_ID, self.geo_transf_IDgeo_transf_ID, self.new_integration_IDnew_integration_ID,
01113         '-iter', self.max_itermax_iter, self.toltol)
01114
01115         # Update class
01116         self.InitializedInitialized = True
01117         self.UpdateStoredDataUpdateStoredData()
01118
01119
01120     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
01121         """
01122         Implementation of the homonym abstract method.
01123         See parent class MemberModel for detailed information.
01124         """
01125         super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec,
01126         def_rec=def_rec, time_rec=time_rec)
01127
01128
01129     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
01130         """
01131         Implementation of the homonym abstract method.
01132         See parent class MemberModel for detailed information.

```

```

01121         """
01122         super().RecordNodeDef(self.iNode_ID, jNode_ID, name_txt, data_dir,
01123                               time_rec=time_rec)
01124
01125     class ForceBasedElementFibersRectRCRectShape(ForceBasedElement):
01126         """
01127         Class that is the children of ForceBasedElement and combine the class FibersRectRCRectShape (fiber
01128         section) to retrieve the information needed.
01129
01130         @param ForceBasedElement: Parent class.
01131         """
01132         def __init__(self, iNode_ID: int, jNode_ID: int, fiber: FibersRectRCRectShape, geo_transf_ID: int,
01133                       new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
01134                       tol=TOL_INTEGRATION, ele_ID = -1):
01135             """
01136             Constructor of the class.
01137
01138             @param iNode_ID (int): ID of the first end node.
01139             @param jNode_ID (int): ID of the second end node.
01140             @param fiber (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
01141             @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
01142             documentation).
01143             @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01144             e.g. computed in ReInit().
01145             @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01146             @param integration_type (str, optional): Integration type. For more information, see
01147             OpenSeesPy documentation.
01148             Defaults to "Lobatto".
01149             @param max_iter (int, optional): Maximal number of iteration to reach the integretion
01150             convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01151             @param tol (float, optional): Tolerance for the integration convergence. Defaults to
01152             TOL_INTEGRATION (Units).
01153             @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01154             IDConvention to define it.
01155             """
01156             self.sectionsection = deepcopy(fiber.section)
01157             super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01158                             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
01159                             max_iter=max_iter, tol=tol, ele_ID= ele_ID)
01160             self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
01161             self.UpdateStoredDataUpdateStoredData()
01162             # Check length
01163             self._CheckL_CheckL()
01164
01165     class ForceBasedElementFibersCircRCCircShape(ForceBasedElement):
01166         """
01167         Class that is the children of ForceBasedElement and combine the class FibersCircRCCircShape (fiber
01168         section) to retrieve the information needed.
01169
01170         @param ForceBasedElement: Parent class.
01171         """
01172         def __init__(self, iNode_ID: int, jNode_ID: int, fiber: FibersCircRCCircShape, geo_transf_ID: int,
01173                       new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
01174                       tol=TOL_INTEGRATION, ele_ID = -1):
01175             """
01176             Constructor of the class.
01177
01178             @param iNode_ID (int): ID of the first end node.
01179             @param jNode_ID (int): ID of the second end node.
01180             @param fiber (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
01181             @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
01182             documentation).
01183             @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01184             e.g. computed in ReInit().
01185             @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01186             @param integration_type (str, optional): Integration type. For more information, see
01187             OpenSeesPy documentation.
01188             Defaults to "Lobatto".
01189             @param max_iter (int, optional): Maximal number of iteration to reach the integretion
01190             convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01191             @param tol (float, optional): Tolerance for the integration convergence. Defaults to
01192             TOL_INTEGRATION (Units).
01193             @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01194             IDConvention to define it.
01195             """
01196             self.sectionsection = deepcopy(fiber.section)
01197             super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01198                             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
01199                             max_iter=max_iter, tol=tol, ele_ID=ele_ID)
01200             self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
01201             self.UpdateStoredDataUpdateStoredData()
01202             # Check length
01203             self._CheckL_CheckL()

```

```

01189 class ForceBasedElementFibersIShapeSteelIShape(ForceBasedElement):
01190     """
01191     Class that is the children of ForceBasedElement and combine the class FibersIShapeSteelIShape
01192     (fiber section) to retrieve the information needed.
01193     @param ForceBasedElement: Parent class.
01194     """
01195     def __init__(self, iNode_ID: int, jNode_ID: int, fiber: FibersIShapeSteelIShape, geo_transf_ID:
01196     int,
01197         new_integration_ID=-1, Ip=5, integration_type="Lobatto", max_iter=MAX_ITER_INTEGRATION,
01198         tol=TOL_INTEGRATION, ele_ID = -1):
01199         """
01200         Constructor of the class.
01201         @param iNode_ID (int): ID of the first end node.
01202         @param jNode_ID (int): ID of the second end node.
01203         @param fiber (FibersIShapeSteelIShape): FibersIShapeSteelIShape fiber section object.
01204         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
01205         documentation).
01206         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01207         e.g. computed in ReInit().
01208         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01209         @param integration_type (str, optional): Integration type. For more information, see
01210         OpenSeesPy documentation.
01211         Defaults to "Lobatto".
01212         @param max_iter (int, optional): Maximal number of iteration to reach the integration
01213         convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01214         @param tol (float, optional): Tolerance for the integration convergence. Defaults to
01215         TOL_INTEGRATION (Units).
01216         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01217         IDConvention to define it.
01218         """
01219         self.sectionsection = deepcopy(fiber.section)
01220         super().__init__(iNode_ID, jNode_ID, fiber.ID, geo_transf_ID,
01221             new_integration_ID=new_integration_ID, Ip=Ip, integration_type=integration_type,
01222             max_iter=max_iter, tol=tol, ele_ID=ele_ID)
01223         self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
01224         self.UpdateStoredDataUpdateStoredData()
01225         # Check length
01226         self._CheckL_CheckL()
01227
01228 class GIFBElement(MemberModel):
01229     """
01230     Class that handles the storage and manipulation of a Gradient-Inelastic Flexibility-based
01231     element's information
01232     (mechanical and geometrical parameters, etc) and the initialisation in the model.
01233     The integration technique is Simpson. For more information, see Sideris and Salehi 2016, 2017 and
01234     2020.
01235     @param MemberModel: Parent abstract class.
01236     """
01237     def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, DBars, fy, geo_transf_ID: int,
01238         lambda_i = -1, lambda_j = -1, Lp = -1, Ip = -1, new_integration_ID = -1,
01239         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01240         ele_ID = -1):
01241         """
01242         Constructor of the class.
01243         @param iNode_ID (int): ID of the first end node.
01244         @param jNode_ID (int): ID of the second end node.
01245         @param fiber_ID (int): ID of the fiber section.
01246         @param DBars (float): Diameter of the vertical reinforcing bars.
01247         @param fy (float): Yield stress of the reinforcing bars.
01248         @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
01249         documentation).
01250         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
01251         end i (0 = no plastic hinge).
01252         Defaults to -1, e.g. plastic hinge in the end i.
01253         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
01254         end j (0 = no plastic hinge).
01255         Defaults to -1, e.g. plastic hinge in the end j.
01256         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01257         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01258         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01259         e.g. computed in ReInit().
01260         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
01261         to TOL_INTEGRATION (Units).
01262         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
01263         to TOL_INTEGRATION*1e4.
01264         @param max_iter (int, optional): Maximal number of iteration to reach the integration
01265         convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01266         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01267         IDConvention to define it.
01268         """
01269         @exception NegativeValue: ID needs to be a positive integer.
01270         @exception NegativeValue: ID needs to be a positive integer.

```



```

01255         @exception NegativeValue: ID needs to be a positive integer.
01256         @exception NegativeValue: D_bars needs to be positive.
01257         @exception NegativeValue: fy needs to be positive.
01258         @exception NegativeValue: ID needs to be a positive integer.
01259         @exception NegativeValue: lambda_i needs to be positive.
01260         @exception NegativeValue: lambda_j needs to be positive.
01261         @exception NegativeValue: No plastic length defined.
01262         @exception NegativeValue: Lp needs to be positive, if different from -1.
01263         @exception NegativeValue: Ip needs to be a positive integer bigger than 3, if different from
-1.
01264         @exception NegativeValue: ID needs to be a positive integer.
01265         @exception NegativeValue: min_tol needs to be positive.
01266         @exception NegativeValue: max_tol needs to be positive.
01267         @exception NegativeValue: max_iter needs to be a positive integer.
01268         @exception NegativeValue: ID needs to be a positive integer, if different from -1.
01269         """
01270         # Check
01271         if iNode_ID < 1: raise NegativeValue()
01272         if jNode_ID < 1: raise NegativeValue()
01273         if fiber_ID < 1: raise NegativeValue()
01274         if D_bars < 0: raise NegativeValue()
01275         if fy < 0: raise NegativeValue()
01276         if geo_transf_ID < 1: raise NegativeValue()
01277         if lambda_i != -1 and lambda_i < 0: raise NegativeValue()
01278         if lambda_j != -1 and lambda_j < 0: raise NegativeValue()
01279         if lambda_i == 0 and lambda_j == 0: print("!!!!!! WARNING !!!!!!! No plastic length defined
for element ID = {}".format(IDConvention(iNode_ID, jNode_ID)))
01280         if Lp != -1 and Lp < 0: raise NegativeValue()
01281         if Ip != -1 and Ip < 3: raise NegativeValue()
01282         if new_integration_ID != -1 and new_integration_ID < 1: raise NegativeValue()
01283         if min_tol < 0: raise NegativeValue()
01284         if max_tol < 0: raise NegativeValue()
01285         if max_iter < 0: raise NegativeValue()
01286         if ele_ID != -1 and ele_ID < 0: raise NegativeValue()
01287
01288         # Arguments
01289         self.iNode_IDiNode_ID = iNode_ID
01290         self.jNode_IDjNode_ID = jNode_ID
01291         self.D_barsD_bars = D_bars
01292         self.fyfy = fy
01293         self.geo_transf_IDgeo_transf_ID = geo_transf_ID
01294         self.fiber_IDfiber_ID = fiber_ID
01295         self.min_tolmin_tol = min_tol
01296         self.max_tolmax_tol = max_tol
01297         self.max_itermax_iter = max_iter
01298
01299         # Initialized the parameters that are dependent from others
01300         self.section_name_tagsection_name_tag = "None"
01301         self.InitializedInitialized = False
01302         self.ReInitReInit(lambda_i, lambda_j, Lp, Ip, new_integration_ID, ele_ID)
01303
01304     def ReInit(self, lambda_i = -1, lambda_j = -1, Lp = -1, Ip = 5, new_integration_ID = -1, ele_ID =
-1):
01305         """
01306         Implementation of the homonym abstract method.
01307         See parent class DataManagement for detailed information.
01308
01309         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01310         Defaults to -1, e.g. plastic hinge in the end i.
01311         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01312         Defaults to -1, e.g. plastic hinge in the end j.
01313         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed here.
01314         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01315         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01316         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01317         """
01318         # Precompute some members
01319         iNode = np.array(nodeCoord(self.iNode_IDiNode_ID))
01320         jNode = np.array(nodeCoord(self.jNode_IDjNode_ID))
01321         self.LL = np.linalg.norm(iNode-jNode)
01322         self.element_IDelement_ID = IDConvention(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID) if
ele_ID == -1 else ele_ID
01323
01324         # Arguments
01325         self.LpLp = self.ComputeLpComputeLp() if Lp == -1 else Lp
01326         self.IpIp = self.ComputeIpComputeIp() if Ip == -1 else Ip
01327         self.lambda_ilambda_i = self.LpLp/self.LL if lambda_i == -1 else lambda_i
01328         self.lambda_jlambda_j = self.LpLp/self.LL if lambda_j == -1 else lambda_j
01329         self.new_integration_IDnew_integration_ID = self.element_IDelement_ID if new_integration_ID ==
-1 else new_integration_ID
01330
01331         # Members
01332         if self.section_name_tagsection_name_tag != "None": self.section_name_tagsection_name_tag =

```



```

self.section_name_tagsection_name_tag + " (modified)"
01333
01334     # Data storage for loading/saving
01335     self.UpdateStoredDataUpdateStoredData()
01336
01337
01338     # Methods
01339     def UpdateStoredData(self):
01340         """
01341         Implementation of the homonym abstract method.
01342         See parent class DataManagement for detailed information.
01343         """
01344         self.data = [{"INFO_TYPE": "GIFBElement"}, # Tag for differentiating different data
01345                     [{"element_ID": self.element_IDelement_ID,
01346                      ["section_name_tag", self.section_name_tagsection_name_tag],
01347                      ["L", self.LL],
01348                      ["D_bars", self.D_barsD_bars],
01349                      ["fy", self.fyfy],
01350                      ["Lp", self.LpLp],
01351                      ["Ip", self.IpIp],
01352                      ["iNode_ID", self.iNode_IDiNode_ID],
01353                      ["lambda_i", self.lambda_ilambda_i],
01354                      ["jNode_ID", self.jNode_IDjNode_ID],
01355                      ["lambda_j", self.lambda_jlambda_j],
01356                      ["fiber_ID", self.fiber_IDfiber_ID],
01357                      ["new_integration_ID", self.new_integration_IDnew_integration_ID],
01358                      ["min_tol", self.min_tolmin_tol],
01359                      ["max_tol", self.max_tolmax_tol],
01360                      ["max_iter", self.max_itermax_iter],
01361                      ["transf_ID", self.geo_transf_IDgeo_transf_ID],
01362                      ["Initialized", self.InitializedInitialized]]}]
01363
01364
01365     def ShowInfo(self, plot = False, block = False):
01366         """
01367         Implementation of the homonym abstract method.
01368         See parent class DataManagement for detailed information.
01369
01370         @param plot (bool, optional): Option to show the plot of the material model. Defaults to
False.
01371         @param block (bool, optional): Option to wait the user command 'plt.show()' (avoiding the stop
of the program everytime that a plot should pop up). Defaults to False.
01372         """
01373         print("")
01374         print("Requested info for GIFBElement member model, ID =
{}".format(self.element_IDelement_ID))
01375         print("Fiber associated, ID = {}".format(self.fiber_IDfiber_ID))
01376         print("Integration type 'Simpson', ID = {}".format(self.new_integration_IDnew_integration_ID))
01377         print("Section associated {}".format(self.section_name_tagsection_name_tag))
01378         print("Length L = {} m".format(self.LL/m_unit))
01379         print("Diameter of the reinforcing bars D_bars = {} mm2".format(self.D_barsD_bars/mm2_unit))
01380         print("Reinforcing bar steel strength fy = {} MPa".format(self.fyfy/MPa_unit))
01381         print("Plastic length Lp = {} mm".format(self.LpLp/mm_unit))
01382         print("Number of integration points along the element Ip = {}, max iter = {}, (min, max tol) =
({}, {})".format(self.IpIp, self.max_itermax_iter, self.min_tolmin_tol, self.max_tolmax_tol))
01383         print("Lambda_i = {} and lambda_j = {}".format(self.lambda_ilambda_i, self.lambda_jlambda_j))
01384         print("Geometric transformation = {}".format(self.geo_transf_IDgeo_transf_ID))
01385         print("")
01386
01387         if plot:
01388             if self.InitializedInitialized:
01389                 plot_member(self.element_arrayelement_array, "GIFB Element, ID =
{}".format(self.element_IDelement_ID))
01390                 if block:
01391                     plt.show()
01392             else:
01393                 print("The GIFBElement is not initialized (element not created), ID =
{}".format(self.element_IDelement_ID))
01394
01395
01396     def CreateMember(self):
01397         """
01398         Method that initialises the member by calling the OpenSeesPy commands through various
functions.
01399         """
01400         self.element_arrayelement_array = [[self.element_IDelement_ID, self.iNode_IDiNode_ID,
self.jNode_IDjNode_ID]]
01401
01402         # Define integration type
01403         beamIntegration('Simpson', self.new_integration_IDnew_integration_ID, self.fiber_IDfiber_ID,
self.IpIp)
01404
01405         # Define element TODO: Dr. Salehi: lambda useless
01406         element('gradientInelasticBeamColumn', self.element_IDelement_ID, self.iNode_IDiNode_ID,
self.jNode_IDjNode_ID, self.geo_transf_IDgeo_transf_ID,
self.new_integration_IDnew_integration_ID, self.lambda_ilambda_i, self.lambda_jlambda_j,
self.LpLp, '-iter', self.max_itermax_iter, self.min_tolmin_tol, self.max_tolmax_tol)

```

```

01408
01409     # Update class
01410     self.InitializedInitialized = True
01411     self.UpdateStoredDataUpdateStoredData()
01412
01413
01414     def Record(self, name_txt: str, data_dir: str, force_rec=True, def_rec=True, time_rec=True):
01415         """
01416         Implementation of the homonym abstract method.
01417         See parent class MemberModel for detailed information.
01418         """
01419         super().Record(self.element_IDelement_ID, name_txt, data_dir, force_rec=force_rec,
01420         def_rec=def_rec, time_rec=time_rec)
01421
01422     def RecordNodeDef(self, name_txt: str, data_dir: str, time_rec=True):
01423         """
01424         Implementation of the homonym abstract method.
01425         See parent class MemberModel for detailed information.
01426         """
01427         super().RecordNodeDef(self.iNode_IDiNode_ID, self.jNode_IDjNode_ID, name_txt, data_dir,
01428         time_rec=time_rec)
01429
01430     def ComputeLp(self):
01431         """
01432         Method that computes the plastic length using Paulay 1992.
01433
01434         @returns double: Plastic length
01435         """
01436         return (0.08*self.LL/m_unit + 0.022*self.D_barsD_bars/m_unit*self.fyfy/MPa_unit)*m_unit
01437
01438
01439     def ComputeIp(self):
01440         """
01441         Compute the number of integration points with equal distance along the element. For more
01442         information, see Salehi and Sideris 2020.
01443
01444         @returns int: Number of integration points
01445         """
01446         tmp = math.ceil(1.5*self.LL/self.LpLp + 1)
01447         if (tmp % 2) == 0:
01448             return tmp + 1
01449         else:
01450             return tmp
01451
01452 class GIFBElementRCRectShape(GIFBElement):
01453     """
01454     Class that is the children of GIFBElement and combine the class RCRectShape (section) to retrieve
01455     the information needed.
01456
01457     @param GIFBElement: Parent class.
01458     """
01459     def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, section: RCRectShape,
01460     geo_transf_ID: int,
01461     lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
01462     min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01463     ele_ID = -1):
01464         """
01465         Constructor of the class.
01466
01467         @param iNode_ID (int): ID of the first end node.
01468         @param jNode_ID (int): ID of the second end node.
01469         @param fiber_ID (int): ID of the fiber section.
01470         @param section (RCRectShape): RCRectShape section object.
01471         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
01472         documentation).
01473         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
01474         end i (0 = no plastic hinge).
01475         Defaults to -1, e.g. plastic hinge in the end i.
01476         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
01477         end j (0 = no plastic hinge).
01478         Defaults to -1, e.g. plastic hinge in the end j.
01479         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01480         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01481         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01482         e.g. computed in ReInit().
01483         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
01484         to TOL_INTEGRATION (Units).
01485         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
01486         to TOL_INTEGRATION*1e4.
01487         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
01488         convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01489         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01490         IDConvention to define it.
01491         """

```

```

01481         self.sectionsection = deepcopy(section)
01482         super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
01483             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01484             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01485         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
01486         self.UpdateStoredDataUpdateStoredData()
01487         # Check length
01488         self._CheckL_CheckL()
01489
01490
01491 class GIFBElementFibersRectRCRectShape(GIFBElement):
01492     """
01493     Class that is the children of GIFBElement and combine the class FibersRectRCRectShape (fiber
01494     section) to retrieve the information needed.
01495
01496     @param GIFBElement: Parent class.
01497     """
01497     def __init__(self, iNode_ID: int, jNode_ID: int, fib: FibersRectRCRectShape, geo_transf_ID: int,
01498         lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
01499         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01500         ele_ID = -1):
01501         """
01502         Constructor of the class.
01503
01504         @param iNode_ID (int): ID of the first end node.
01505         @param jNode_ID (int): ID of the second end node.
01506         @param fib (FibersRectRCRectShape): FibersRectRCRectShape fiber section object.
01507         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
01508         documentation).
01509         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
01510         end i (0 = no plastic hinge).
01511         Defaults to -1, e.g. plastic hinge in the end i.
01512         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
01513         end j (0 = no plastic hinge).
01514         Defaults to -1, e.g. plastic hinge in the end j.
01515         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01516         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01517         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
01518         e.g. computed in ReInit().
01519         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
01520         to TOL_INTEGRATION (Units).
01521         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
01522         to TOL_INTEGRATION*1e4.
01523         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
01524         convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01525         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
01526         IDConvention to define it.
01527         """
01528         self.sectionsection = deepcopy(fib.section)
01529         super().__init__(iNode_ID, jNode_ID, fib.ID, self.sectionsection.D_bars,
01530             self.sectionsection.fy, geo_transf_ID,
01531             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01532             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01533         self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
01534         self.UpdateStoredDataUpdateStoredData()
01535         # Check length
01536         self._CheckL_CheckL()
01537
01538
01539 class GIFBElementRCCircShape(GIFBElement):
01540     """
01541     Class that is the children of GIFBElement and combine the class RCCircShape (section) to retrieve
01542     the information needed.
01543
01544     @param GIFBElement: Parent class.
01545     """
01546     def __init__(self, iNode_ID: int, jNode_ID: int, fiber_ID: int, section: RCCircShape,
01547         geo_transf_ID: int,
01548         lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
01549         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01550         ele_ID = -1):
01551         """
01552         Constructor of the class.
01553
01554         @param iNode_ID (int): ID of the first end node.
01555         @param jNode_ID (int): ID of the second end node.
01556         @param fiber_ID (int): ID of the fiber section.
01557         @param section (RCCircShape): RCCircShape section object.
01558         @param geo_transf_ID (int): The geometric transformation (for more information, see OpenSeesPy
01559         documentation).
01560         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
01561         end i (0 = no plastic hinge).
01562         Defaults to -1, e.g. plastic hinge in the end i.
01563         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
01564         end j (0 = no plastic hinge).
01565         Defaults to -1, e.g. plastic hinge in the end j.
01566         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().

```

```

01551         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01552         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01553         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION (Units).
01554         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION*1e4.
01555         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01556         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01557         """
01558         self.sectionsection = deepcopy(section)
01559         super().__init__(iNode_ID, jNode_ID, fiber_ID, section.D_bars, section.fy, geo_transf_ID,
01560             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01561             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01562         self.section_name_tagsection_name_tagsection_name_tag = section.name_tag
01563         self.UpdateStoredDataUpdateStoredData()
01564         # Check length
01565         self._CheckL_CheckL()
01566
01567
01568 class GIBFElementFibersCircRCCircShape(GIBFElement):
01569     """
01570     Class that is the children of GIBFElement and combine the class FibersCircRCCircShape (fiber
section) to retrieve the information needed.
01571
01572     @param GIBFElement: Parent class.
01573     """
01574     def __init__(self, iNode_ID: int, jNode_ID: int, fib: FibersCircRCCircShape, geo_transf_ID: int,
01575         lambda_i=-1, lambda_j=-1, Lp=-1, Ip=-1, new_integration_ID=-1,
01576         min_tol = TOL_INTEGRATION, max_tol = TOL_INTEGRATION*1e4, max_iter = MAX_ITER_INTEGRATION,
01577         ele_ID = -1):
01578         """
01579         Constructor of the class.
01580
01581         @param iNode_ID (int): ID of the first end node.
01582         @param jNode_ID (int): ID of the second end node.
01583         @param fib (FibersCircRCCircShape): FibersCircRCCircShape fiber section object.
01584         @param geo_transf_ID (int): A geometric transformation (for more information, see OpenSeesPy
documentation).
01585         @param lambda_i (float, optional): Fraction of beam length over the plastic hinge length at
end i (0 = no plastic hinge).
01586         Defaults to -1, e.g. plastic hinge in the end i.
01587         @param lambda_j (float, optional): Fraction of beam length over the plastic hinge length at
end j (0 = no plastic hinge).
01588         Defaults to -1, e.g. plastic hinge in the end j.
01589         @param Lp (float, optional): Plastic hinge length. Defaults to -1, e.g. computed in ReInit().
01590         @param Ip (int, optional): Number of integration points (min. 3). Defaults to 5.
01591         @param new_integration_ID (int, optional): ID of the integration technique. Defaults to -1,
e.g. computed in ReInit().
01592         @param min_tol (float, optional): Minimal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION (Units).
01593         @param max_tol (float, optional): Maximal tolerance for the integration convergence. Defaults
to TOL_INTEGRATION*1e4.
01594         @param max_iter (int, optional): Maximal number of iteration to reach the integretion
convergence. Defaults to MAX_ITER_INTEGRATION (Units).
01595         @param ele_ID (int, optional): Optional ID of the element. Defaults to -1, e.g. use
IDConvention to define it.
01596         """
01597         self.sectionsection = deepcopy(fib.section)
01598         super().__init__(iNode_ID, jNode_ID, fib.ID, self.sectionsection.D_bars,
self.sectionsection.fy, geo_transf_ID,
01599             lambda_i=lambda_i, lambda_j=lambda_j, Lp=Lp, Ip=Ip, new_integration_ID=new_integration_ID,
01600             min_tol=min_tol, max_tol=max_tol, max_iter=max_iter, ele_ID = ele_ID)
01601         self.section_name_tagsection_name_tagsection_name_tag = self.sectionsection.name_tag
01602         self.UpdateStoredDataUpdateStoredData()
01603         # Check length
01604         self._CheckL_CheckL()
01605
01606
01607

```

## 8.21 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/README.md File Reference

## 8.22 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Section.py File Reference

### Classes

- class [RCircShape](#)  
*Class that stores functions, geometric and mechanical properties of RC circular shape profile.*
- class [RCRectShape](#)  
*Class that stores functions, geometric and mechanical properties of RC rectangular shape profile.*
- class [RCSquareShape](#)  
*Class that is the children of [RCRectShape](#) and cover the specific case of square RC sections.*
- class [Section](#)  
*Parent abstract class for the storage and manipulation of a section's information (mechanical and geometrical parameters, etc).*
- class [SteelShape](#)  
*Class that stores functions, geometric and mechanical properties of a steel double symmetric I-shape profile.*

### Namespaces

- namespace [Section](#)  
*Module for the section (steel I shape profiles, RC circular/square/rectangular sections).*

### Functions

- def [ComputeACircle](#) (D)  
*Function that computes the area of one circle (reinforcing bar or hoop).*
- def [ComputeRho](#) (A, nr, A\_tot)  
*Compute the ratio of area of a reinforcement to area of a section.*

## 8.23 Section.py

[Go to the documentation of this file.](#)

```
00001 """
00002 Module for the section (steel I shape profiles, RC circular/square/rectangular sections).
00003 Carmine Schipani, 2021
00004 """
00005
00006 import numpy as np
00007 import math
00008 from copy import copy, deepcopy
00009 from OpenSeesPyAssistant.DataManagement import *
00010 from OpenSeesPyAssistant.ErrorHandling import *
00011 from OpenSeesPyAssistant.Units import *
00012
00013 class Section(DataManagement):
00014     """
00015     Parent abstract class for the storage and manipulation of a section's information (mechanical and
00016     geometrical parameters, etc).
00017
00017     @param DataManagement: Parent abstract class.
00018     """
```

```

00019     pass
00020
00021
00022 class SteelIShape(Section):
00023     """
00024     Class that stores functions, geometric and mechanical properties of a steel double symmetric
    I-shape profile.
00025     The parameter 'n' is used as global throughout the SteelIShape sections to optimise the program
    (given the fact that is constant everytime).
00026
00027     @param Section: Parent abstract class.
00028     """
00029     global n
00030     n = 10.0
00031
00032     def __init__(self, Type: str, d, bf, tf, tw, L, r, E, Fy, Fy_web = -1, name_tag = "Not Defined"):
00033         """
00034         The constructor of the class.
00035
00036         @param Type (str): Type of the section. It can be 'Col' for column or 'Beam' for beams.
00037         @param d (float): Depth of the section.
00038         @param bf (float): Flange's width of the section
00039         @param tf (float): Flange's thickness of the section
00040         @param tw (float): Web's thickness of the section
00041         @param L (float): Effective length of the element associated with this section.
00042             If the panel zone is present, exclude its dimension.
00043         @param r (float): Radius of the weld fillets of the section.
00044         @param E (float): Young modulus of the section.
00045         @param Fy (float): Yield strength of the flange of the section. Used as the yield strength of
    the entire section.
00046         @param Fy_web (float, optional): Yield strength of the web of the section. Used for panel zone
    associated to this section.
00047             Defaults to -1, e.g. computed in __init__() as equal to Fy.
00048         @param name_tag (str, optional): Name TAG of the section. Defaults to "Not Defined".
00049
00050         @exception WrongArgument: Type needs to be 'Col' or 'Beam'.
00051         @exception NegativeValue: d needs to be positive.
00052         @exception NegativeValue: bf needs to be positive.
00053         @exception NegativeValue: tf needs to be positive.
00054         @exception NegativeValue: tw needs to be positive.
00055         @exception NegativeValue: L needs to be positive.
00056         @exception NegativeValue: r needs to be positive.
00057         @exception NegativeValue: E needs to be positive.
00058         @exception NegativeValue: Fy needs to be positive.
00059         @exception NegativeValue: Fy_web needs to be positive if different from -1.
00060         @exception InconsistentGeometry: tw should be smaller than bf.
00061         @exception InconsistentGeometry: tf needs to be smaller than half of d
00062         @exception InconsistentGeometry: r should be less than half bf and d
00063         """
00064         # Check
00065         if Type != "Beam" and Type != "Col": raise WrongArgument()
00066         if d < 0: raise NegativeValue()
00067         if bf < 0: raise NegativeValue()
00068         if tf < 0: raise NegativeValue()
00069         if tw < 0: raise NegativeValue()
00070         if L < 0: raise NegativeValue()
00071         if r < 0: raise NegativeValue()
00072         if E < 0: raise NegativeValue()
00073         if Fy < 0: raise NegativeValue()
00074         if Fy_web != -1 and Fy_web < 0: raise NegativeValue()
00075         if tw > bf: raise InconsistentGeometry()
00076         if tf > d/2: raise InconsistentGeometry()
00077         if r > bf/2 or r > d/2: raise InconsistentGeometry()
00078
00079         # Arguments
00080         self.TypeType = Type
00081         self.dd = d
00082         self.bfbf = bf
00083         self.tftf = tf
00084         self.twtw = tw
00085         self.LL = L
00086         self.rr = r
00087         self.EE = E
00088         self.FyFy = Fy
00089         self.Fy_webFy_web = Fy if Fy_web == -1 else Fy_web
00090         self.name_tagname_tag = name_tag
00091
00092         # Initialized the parameters that are dependent from others
00093         self.ReInitReInit()
00094
00095     def ReInit(self):
00096         """
00097         Implementation of the homonym abstract method.
00098         See parent class DataManagement for detailed information.
00099         """
00100         # Member
00101         self.h_lh_1 = self.dd - 2.0*self.rr - 2.0*self.tftf

```

```

00102         self.AA = self.ComputeAComputeA()
00103         self.NplNpl = self.AA*self.FyFy
00104         self.IyIy = self.ComputeIyComputeIy()
00105         self.IzIz = self.ComputeIzComputeIz()
00106         self.WplyWply = self.ComputeWplyComputeWply()
00107         self.WplzWplz = self.ComputeWplzComputeWplz()
00108         self.MyMy = self.FyFy*self.WplyWply
00109         self.Iy_modIy_mod = self.IyIy*(n + 1.0)/n
00110         self.iziz = self.Compute_izCompute_iz()
00111         self.iyiy = self.Compute_iyCompute_iy()
00112
00113         # Data storage for loading/saving
00114         self.UpdateStoredDataUpdateStoredData()
00115
00116     def UpdateStoredData(self):
00117         """
00118         Implementation of the homonym abstract method.
00119         See parent class DataManagement for detailed information.
00120         """
00121         self.datadata = [{"INFO_TYPE", "SteelIShape"}, # Tag for differentiating different data
00122                           [{"name_tag", self.name_tagname_tag},
00123                            ["Type", self.TypeType],
00124                            ["d", self.dd],
00125                            ["bf", self.bfbf],
00126                            ["tf", self.tftf],
00127                            ["tw", self.twtw],
00128                            ["L", self.LL],
00129                            ["r", self.rr],
00130                            ["h_l", self.h_lh_l],
00131                            ["E", self.EE],
00132                            ["Fy", self.FyFy],
00133                            ["Fy_web", self.Fy_webFy_web],
00134                            ["A", self.AA],
00135                            ["Iy", self.IyIy],
00136                            ["Iz", self.IzIz],
00137                            ["Wply", self.WplyWply],
00138                            ["Wplz", self.WplzWplz],
00139                            ["Iy_mod", self.Iy_modIy_mod],
00140                            ["iy", self.iyiy],
00141                            ["iz", self.iziz],
00142                            ["Npl", self.NplNpl],
00143                            ["My", self.MyMy]]]
00144
00145     def ShowInfo(self):
00146         """
00147         Implementation of the homonym abstract method.
00148         See parent class DataManagement for detailed information.
00149         """
00150         print("")
00151         print("Requested info for steel I shape section of type = {} and name tag = {}".format(self.TypeType, self.name_tagname_tag))
00152         print("d = {} mm".format(self.dd/mm_unit))
00153         print("Fy = {} MPa".format(self.FyFy/MPa_unit))
00154         print("Fy web = {} MPa".format(self.Fy_webFy_web/MPa_unit))
00155         print("E = {} GPa".format(self.EE/GPa_unit))
00156         print("h_l = {} mm".format(self.h_lh_l/mm_unit))
00157         print("A = {} mm2".format(self.AA/mm2_unit))
00158         print("Iy = {} mm4".format(self.IyIy/mm4_unit))
00159         print("Iz = {} mm4".format(self.IzIz/mm4_unit))
00160         print("Wply = {} mm3".format(self.WplyWply/mm3_unit))
00161         print("Wplz = {} mm3".format(self.WplzWplz/mm3_unit))
00162         print("Iy_mod = {} mm4".format(self.Iy_modIy_mod/mm4_unit))
00163         print("iy = {} mm".format(self.iyiy/mm_unit))
00164         print("iz = {} mm".format(self.iziz/mm_unit))
00165         print("My = {} kNm".format(self.MyMy/kNm_unit))
00166         print("Npl = {} kN".format(self.NplNpl/kN_unit))
00167         print("")
00168
00169
00170     def ComputeA(self):
00171         """
00172         Compute the area of a double symmetric I-profile section with fillets.
00173
00174         @returns float: Area of the I shape section (with fillets included)
00175         """
00176         # d : The depth
00177         # bf : The flange's width
00178         # tf : The flange's thickness
00179         # tw : The web's thickness
00180         # r : The weld fillet radius
00181
00182         # without fillets bf*tf*2 + tw*(d-2*tf)
00183         return 2.0*self.bfbf*self.tftf+self.twtw*(self.dd-2.0*self.tftf)+0.8584*self.rr**2
00184
00185     def ComputeIy(self):
00186         """
00187         Compute the moment of inertia of a double symmetric I-profile section, with respect to its

```

```

    strong axis with fillets.
00188
00189     @returns float: The moment of inertia with respect to the strong axis.
00190     """
00191     # d :      The depth
00192     # bf :     The flange's width
00193     # tf :     The flange's thickness
00194     # tw :     The web's thickness
00195     # r :      The weld fillet radius
00196
00197     # without fillets: bf*tf/2*(d-tf)**2 + bf*tf**3/6 + (d-tf*2)**3*tf/12
00198     return
00199     (self.bfbf*self.dd**3.0-(self.bfbf-self.twtw)*(self.dd-2.0*self.tftf)**3)/12.0+0.8584*self.rr**2*(0.5*self.dd-self.tftf)
00200
00201     def ComputeIz(self):
00202         """
00203         Compute the moment of inertia of a double symmetric I-profile section, with respect to its
00204         weak axis with fillets.
00205
00206         @returns float: The moment of inertia with respect to the weak axis.
00207         """
00208         # d :      The depth
00209         # bf :     The flange's width
00210         # tf :     The flange's thickness
00211         # tw :     The web's thickness
00212         # r :      The weld fillet radius
00213
00214         return
00215         (self.tftf*self.bfbf**3)/6.0+((self.dd-2.0*self.tftf)*self.twtw**3)/12.0+0.8584*self.rr**2*(0.5*self.twtw+0.2234*self.rr)
00216
00217     def ComputeWply(self):
00218         """
00219         Compute the plastic modulus of a double symmetric I-profile section, with respect to its
00220         strong axis with fillets.
00221
00222         @returns float: The plastic modulus with respect to the strong axis.
00223         """
00224         # d :      The depth
00225         # bf :     The flange's width
00226         # tf :     The flange's thickness
00227         # tw :     The web's thickness
00228         # r :      The weld fillet radius
00229
00230         return
00231         self.bfbf*self.tftf*(self.dd-self.tftf)+(self.dd-2.0*self.tftf)**2.0*(self.twtw/4.0)+0.4292*self.rr**2*(self.dd-2.0*self.tftf)
00232
00233     def ComputeWplz(self):
00234         """
00235         Compute the plastic modulus of a double symmetric I-profile section, with respect to its weak
00236         axis with fillets.
00237
00238         @returns float: The plastic modulus with respect to the weak axis.
00239         """
00240         # d :      The depth
00241         # bf :     The flange's width
00242         # tf :     The flange's thickness
00243         # tw :     The web's thickness
00244         # r :      The weld fillet radius
00245
00246         return
00247         (self.tftf*self.bfbf**2)/2+(self.dd-2.0*self.tftf)*(self.twtw**2/4.0)+0.4292*self.rr**2*(self.twtw+0.4467*self.rr)
00248
00249     def Compute_iy(self):
00250         """
00251         Compute the gyration radius with respect to the strong axis.
00252
00253         @returns float: The gyration radius with respect to the strong axis.
00254         """
00255         # Iy :     The second moment of inertia with respect to thte strong axis
00256         # A :      The area
00257
00258         return math.sqrt(self.IyIy/self.AA)
00259
00260     def Compute_iz(self):
00261         """
00262         Compute the gyration radius with respect to the weak axis.
00263
00264         @returns float: The gyration radius with respect to the weak axis.
00265         """
00266         # Iz :     The second moment of inertia with respect to thte weak axis
00267         # A :      The area
00268
00269         return math.sqrt(self.IzIz/self.AA)
00270
00271 class RCRectShape(Section):
00272     """
00273     Class that stores funcions, geometric and mechanical properties of RC rectangular shape profile.

```



```

00267     Note that for the validity of the formulas, at least one bar per corner and at least one hoop
closed (with 135 degrees possibly).
00268
00269     @param Section: Parent abstract class.
00270     """
00271     def __init__(self, b, d, L, e, fc, D_bars, bars_position_x: np.ndarray, bars_ranges_position_y:
np.ndarray, fy, Ey,
00272         D_hoops, s, fs, Es, name_tag = "Not Defined", rho_s_x = -1, rho_s_y = -1, Ec = -1):
00273         """
00274         The constructor of the class.
00275
00276         @param b (float): Width of the section.
00277         @param d (float): Depth of the section.
00278         @param L (float): Effective length of the element associated with this section.
00279             If the panel zone is present, exclude its dimension.
00280         @param e (float): Concrete cover.
00281         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00282         @param D_bars (float): Diameter of the reinforcing bars.
00283         @param bars_position_x (np.ndarray): Array with a range of aligned vertical reinforcing bars
for each row in x direction.
00284             Distances from border to bar centerline, bar to bar centerlines and
00285             finally bar centerline to border in the x direction (aligned).
00286             Starting from the left to right, from the top range to the bottom one.
00287             The number of bars for each range can vary; in this case, add this argument when defining
the array " dtype = object".
00288         @param bars_ranges_position_y (np.ndarray): Array of dimension 1 with the position or spacing
in y of the ranges in bars_position_x.
00289             Distances from border to range centerlines, range to range centerlines and
00290             finally range centerline to border in the y direction.
00291             Starting from the top range to the bottom one.
00292         @param fy (float): Yield stress for reinforcing bars.
00293         @param Ey (float): Young modulus for reinforcing bars.
00294         @param D_hoops (float): Diameter of the hoops.
00295         @param s (float): Centerline distance for the hoops.
00296         @param fs (float): Yield stress for the hoops.
00297         @param Es (float): Young modulus for the hoops
00298         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00299         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the x direction.
00300             Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00301         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the y direction.
00302             Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00303         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in
__init__() and ReInit().
00304
00305         @exception NegativeValue: b needs to be positive.
00306         @exception NegativeValue: d needs to be positive.
00307         @exception NegativeValue: L needs to be positive.
00308         @exception NegativeValue: e needs to be positive.
00309         @exception PositiveValue: fc needs to be negative.
00310         @exception NegativeValue: D_bars needs to be positive.
00311         @exception NegativeValue: fy needs to be positive.
00312         @exception NegativeValue: Ey needs to be positive.
00313         @exception NegativeValue: D_hoops needs to be positive.
00314         @exception NegativeValue: s needs to be positive.
00315         @exception NegativeValue: fs needs to be positive.
00316         @exception NegativeValue: Es needs to be positive.
00317         @exception NegativeValue: rho_s_x needs to be positive if different from -1.
00318         @exception NegativeValue: rho_s_y needs to be positive if different from -1.
00319         @exception NegativeValue: Ec needs to be positive if different from -1.
00320         @exception WrongDimension: Number of lists in the list bars_position_x needs to be the same of
the length of bars_ranges_position_y - 1.
00321         @exception InconsistentGeometry: The sum of the distances for each list in bars_position_x
should be equal to the section's width (tol = 5 mm).
00322         @exception InconsistentGeometry: The sum of the distances in bars_ranges_position_y should be
equal to the section's depth (tol = 5 mm).
00323         @exception InconsistentGeometry: e should be smaller than half the depth and the width of the
section.
00324         """
00325         # Check
00326         if b < 0: raise NegativeValue()
00327         if d < 0: raise NegativeValue()
00328         if L < 0: raise NegativeValue()
00329         if e < 0: raise NegativeValue()
00330         if fc > 0: raise PositiveValue()
00331         if D_bars < 0: raise NegativeValue()
00332         if fy < 0: raise NegativeValue()
00333         if Ey < 0: raise NegativeValue()
00334         if D_hoops < 0: raise NegativeValue()
00335         if s < 0: raise NegativeValue()
00336         if fs < 0: raise NegativeValue()
00337         if Es < 0: raise NegativeValue()
00338         if rho_s_x != -1 and rho_s_x < 0: raise NegativeValue()
00339         if rho_s_y != -1 and rho_s_y < 0: raise NegativeValue()
00340         if Ec != -1 and Ec < 0: raise NegativeValue()
00341         if np.size(bars_position_x) != np.size(bars_ranges_position_y)-1: raise WrongDimension()

```

```

00342         geometry_tol = 5*mm_unit
00343         for bars in bars_position_x:
00344             if abs(np.sum(bars) - b) > geometry_tol: raise InconsistentGeometry()
00345         if abs(np.sum(bars_ranges_position_y)-d) > geometry_tol: raise InconsistentGeometry()
00346         if e > b/2 or e > d/2: raise InconsistentGeometry()
00347         warning_min_bars = "!!!!!!! WARNING !!!!!!! The hypothesis of one bar per corner (aligned) is
not fulfilled."
00348         if len(bars_position_x) < 2:
00349             print(warning_min_bars)
00350         elif len(bars_position_x[0]) < 3 or len(bars_position_x[-1]) < 3:
00351             print(warning_min_bars)
00352
00353         # Arguments
00354         self.bb = b
00355         self.dd = d
00356         self.LL = L
00357         self.ee = e
00358         self.fcfc = fc
00359         self.D_barsD_bars = D_bars
00360         self.bars_position_xbars_position_x = deepcopy(bars_position_x)
00361         self.bars_ranges_position_ybars_ranges_position_y = copy(bars_ranges_position_y)
00362         self.fyfy = fy
00363         self.EyEy = Ey
00364         self.D_hoopsD_hoops = D_hoops
00365         self.ss = s
00366         self.fsfs = fs
00367         self.EsEs = Es
00368         self.name_tagname_tag = name_tag
00369
00370         # Initialized the parameters that are dependent from others
00371         self.ReInitReInit(rho_s_x, rho_s_y, Ec)
00372
00373     def ReInit(self, rho_s_x = -1, rho_s_y = -1, Ec = -1):
00374         """
00375         Implementation of the homonym abstract method.
00376         See parent class DataManagement for detailed information.
00377
00378         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the x direction.
00380             Defaults to -1, e.g. computed assuming one range of hoops.
00381         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the y direction.
00382             Defaults to -1, e.g. computed assuming one range of hoops.
00383         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed
according to Mander et Al. 1988.
00384         """
00385         # Precompute some members
00386         self.cl_hoopscl_hoops = self.ee + self.D_hoopsD_hoops/2.0 # centerline distance from the
border of the extreme confining hoops
00387         self.cl_barscl_bars = self.ee + self.D_barsD_bars/2.0 + self.D_hoopsD_hoops # centerline
distance from the border of the corner bars
00388         self.bcbc = self.bb - self.cl_hoopscl_hoops*2
00389         self.dcdc = self.dd - self.cl_hoopscl_hoops*2
00390         self.AsAs = ComputeACircle(self.D_hoopsD_hoops)
00391
00392         # Arguments
00393         self.rho_s_xrho_s_x = 2.0*ComputeRho(self.AsAs, 1, self.bcbc*self.ss) if rho_s_x == -1 else
rho_s_x
00394         self.rho_s_yrho_s_y = 2.0*ComputeRho(self.AsAs, 1, self.dcdc*self.ss) if rho_s_y == -1 else
rho_s_y
00395         self.EcEc = self.ComputeEcComputeEc() if Ec == -1 else Ec
00396
00397         # Members
00398         self.nr_barsnr_bars = self.ComputeNrBarsComputeNrBars()
00399         self.AA = self.ComputeAComputeA()
00400         self.AcAc = self.ComputeAcComputeAc()
00401         self.AyAy = ComputeACircle(self.D_barsD_bars)
00402         self.rho_barsrho_bars = ComputeRho(self.AyAy, self.nr_barsnr_bars, self.AA)
00403         self.IyIy = self.ComputeIyComputeIy()
00404         self.IzIz = self.ComputeIzComputeIz()
00405
00406         # Data storage for loading/saving
00407         self.UpdateStoredDataUpdateStoredData()
00408
00409     def UpdateStoredData(self):
00410         """
00411         Implementation of the homonym abstract method.
00412         See parent class DataManagement for detailed information.
00413         """
00414         self.datadata = [{"INFO_TYPE", "RCRectShape"}, # Tag for differentiating different data
00415             [{"name_tag", self.name_tagname_tag},
00416             [{"b", self.bb},
00417             [{"d", self.dd},
00418             [{"bc", self.bcbc},
00419             [{"dc", self.dcdc},

```

```

00421         ["L", self.LL],
00422         ["e", self.ee],
00423         ["A", self.AA],
00424         ["Ac", self.AcAc],
00425         ["Iy", self.IyIy],
00426         ["Iz", self.IzIz],
00427         ["fc", self.fcfc],
00428         ["Ec", self.EcEc],
00429         ["D_bars", self.D_barsD_bars],
00430         ["nr_bars", self.nr_barsnr_bars],
00431         ["Ay", self.AyAy],
00432         ["bars_position_x", self.bars_position_xbars_position_x],
00433         ["bars_ranges_position_y", self.bars_ranges_position_ybars_ranges_position_y],
00434         ["rho_bars", self.rho_barsrho_bars],
00435         ["cl_bars", self.cl_barscl_bars],
00436         ["fy", self.fyfy],
00437         ["Ey", self.EyEy],
00438         ["D_hoops", self.D_hoopsD_hoops],
00439         ["s", self.ss],
00440         ["As", self.AsAs],
00441         ["rho_s_x", self.rho_s_xrho_s_x],
00442         ["rho_s_y", self.rho_s_yrho_s_y],
00443         ["cl_hoops", self.cl_hoopscl_hoops],
00444         ["fs", self.fsfs],
00445         ["Es", self.EsEs]]
00446
00447
00448     def ShowInfo(self):
00449         """
00450         Implementation of the homonym abstract method.
00451         See parent class DataManagement for detailed information.
00452         """
00453         print("")
00454         print("Requested info for RC rectangular section of name tag =
00455         {}".format(self.name_tagname_tag))
00456         print("Width of the section b = {} mm".format(self.bb/mm_unit))
00457         print("Depth of the section d = {} mm".format(self.dd/mm_unit))
00458         print("Concrete cover e = {} mm".format(self.ee/mm_unit))
00459         print("Concrete area A = {} mm2".format(self.AA/mm2_unit))
00460         print("Core concrete area Ac = {} mm2".format(self.AcAc/mm2_unit))
00461         print("Unconfined concrete compressive strength fc = {} MPa".format(self.fcfc/MPa_unit))
00462         print("Young modulus for concrete Ec = {} GPa".format(self.EcEc/GPa_unit))
00463         print("Diameter of the reinforcing bars D_bars = {} mm and area of one bar Ay = {} mm2 with {}
00464         bars".format(self.D_barsD_bars/mm_unit, self.AyAy/mm2_unit, self.nr_barsnr_bars))
00465         print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {}
00466         mm2".format(self.D_hoopsD_hoops/mm_unit, self.AsAs/mm2_unit))
00467         print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bars =
00468         {}".format(self.rho_barsrho_bars))
00469         print("Ratio of area of lateral reinforcement to lateral area of concrete section in x rho_s_x
00470         = {}".format(self.rho_s_xrho_s_x))
00471         print("Ratio of area of lateral reinforcement to lateral area of concrete section in y rho_s_y
00472         = {}".format(self.rho_s_yrho_s_y))
00473         print("Moment of inertia of the circular section (strong axis) Iy = {}
00474         mm4".format(self.IyIy/mm4_unit))
00475         print("Moment of inertia of the circular section (weak axis) Iz = {}
00476         mm4".format(self.IzIz/mm4_unit))
00477         print("")
00478
00479     def ComputeNrBars(self):
00480         """
00481         Compute the number of vertical bars in the array bars_position_x (note that this list of lists
00482         can have different list sizes).
00483
00484         @returns int: Number of vertical reinforcing bars.
00485         """
00486         nr_bars = 0
00487         for range in self.bars_position_xbars_position_x:
00488             nr_bars += np.size(range)-1
00489
00490         return nr_bars
00491
00492
00493     def ComputeEc(self):
00494         """
00495         Compute Ec using the formula from Mander et Al. 1988.
00496
00497         @returns float: Young modulus of concrete.
00498         """
00499         return 5000.0 * math.sqrt(-self.fcfc/MPa_unit) * MPa_unit
00500
00501
00502     def ComputeA(self):
00503         """
00504         Compute the area for a rectangular section.
00505         """

```

```

00499         @returns float: Total area.
00500         """
00501         return self.bb * self.dd
00502
00503
00504     def ComputeAc(self):
00505         """
00506         Compute the confined area (area inside the centerline of the hoops, according to Mander et Al.
1988).
00507
00508         @returns float: Confined area.
00509         """
00510         return self.bc * self.dd
00511
00512
00513     def ComputeIy(self):
00514         """
00515         Compute the moment of inertia of the rectangular section with respect to the strong axis.
00516
00517         @returns float: Moment of inertia (strong axis)
00518         """
00519         return self.bb * self.dd**3 / 12.0
00520
00521
00522     def ComputeIz(self):
00523         """
00524         Compute the moment of inertia of the rectangular section with respect to the weak axis.
00525
00526         @returns float: Moment of inertia (weak axis)
00527         """
00528         return self.dd * self.bb**3 / 12.0
00529
00530
00531 class RCSquareShape(RCRectShape):
00532     """
00533     Class that is the children of RCRectShape and cover the specific case of square RC sections.
00534
00535     @param RCRectShape: Parent class.
00536     """
00537     def __init__(self, b, L, e, fc, D_bars, bars_position_x: np.ndarray, bars_ranges_position_y:
np.ndarray, fy, Ey, D_hoops, s, fs, Es, name_tag="Not Defined", rho_s_x=-1, rho_s_y=-1, Ec=-1):
00538         """
00539         Constructor of the class. It passes the arguments into the parent class to generate the
specific case of a square RC section.
00540
00541         @param b (float): Width/depth of the section.
00542         @param L (float): Effective length of the element associated with this section.
00543         If the panel zone is present, exclude its dimension.
00544         @param e (float): Concrete cover.
00545         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00546         @param D_bars (float): Diameter of the reinforcing bars.
00547         @param bars_position_x (np.ndarray): Distances from border to bar centerline, bar to bar
centerlines and
00548         finally bar centerline to border in the x direction (aligned).
00549         Starting from the left to right, from the top range to the bottom one.
00550         The number of bars for each range can vary; in this case, add this argument when defining
the array " dtype = object".
00551         @param bars_ranges_position_y (np.ndarray): Distances from border to range centerlines, range
to range centerlines and
00552         finally range centerline to border in the y direction.
00553         Starting from the top range to the bottom one.
00554         @param fy (float): Yield stress for reinforcing bars.
00555         @param Ey (float): Young modulus for reinforcing bars.
00556         @param D_hoops (float): Diameter of the hoops.
00557         @param s (float): Vertical centerline spacing between hoops.
00558         @param fs (float): Yield stress for the hoops.
00559         @param Es (float): Young modulus for the hoops
00560         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00561         @param rho_s_x (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the x direction.
00562         Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00563         @param rho_s_y (float, optional): Ratio of the transversal area of the hoops to the associated
concrete area in the y direction.
00564         Defaults to -1, e.g. computed in __init__() and ReInit() assuming one range of hoops.
00565         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in
__init__() and ReInit().
00566         """
00567         super().__init__(b, b, L, e, fc, D_bars, bars_position_x, bars_ranges_position_y, fy, Ey,
D_hoops, s, fs, Es, name_tag, rho_s_x, rho_s_y, Ec)
00568
00569
00570 class RCCircShape(Section):
00571     """
00572     Class that stores functions, geometric and mechanical properties of RC circular shape profile.
00573     Note that for the validity of the formulas, the hoops needs to be closed (with 135 degrees
possibly).
00574

```

```

00575     @param Section: Parent abstract class.
00576     """
00577     def __init__(self, b, L, e, fc, D_bars, n_bars: int, fy, Ey, D_hoops, s, fs, Es, name_tag = "Not
Defined", rho_s_vol = -1, Ec = -1):
00578         """
00579         The constructor of the class.
00580
00581         @param b (float): Width of the section.
00582         @param L (float): Effective length of the element associated with this section.
00583             If the panel zone is present, exclude its dimension.
00584         @param e (float): Concrete cover.
00585         @param fc (float): Unconfined concrete compressive strength (cylinder test).
00586         @param D_bars (float): Diameter of the vertical reinforcing bars.
00587         @param n_bars (int): Number of vertical reinforcing bars.
00588         @param fy (float): Yield stress for reinforcing bars.
00589         @param Ey (float): Young modulus for reinforcing bars.
00590         @param D_hoops (float): Diameter of the hoops.
00591         @param s (float): Vertical centerline spacing between hoops.
00592         @param fs (float): Yield stress for the hoops.
00593         @param Es (float): Young modulus for the hoops
00594         @param name_tag (str, optional): A nametag for the section. Defaults to "Not Defined".
00595         @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the
volume of confined concrete core.
00596             Defaults to -1, e.g. computed according to Mander et Al. 1988.
00597         @param Ec (float, optional): Young modulus for concrete. Defaults to -1, e.g. computed in
__init__() and ReInit().
00598
00599         @exception NegativeValue: b needs to be positive.
00600         @exception NegativeValue: L needs to be positive.
00601         @exception NegativeValue: e needs to be positive.
00602         @exception PositiveValue: fc needs to be negative.
00603         @exception NegativeValue: D_bars needs to be positive.
00604         @exception NegativeValue: n_bars needs to be a positive integer.
00605         @exception NegativeValue: fy needs to be positive.
00606         @exception NegativeValue: Ey needs to be positive.
00607         @exception NegativeValue: D_hoops needs to be positive.
00608         @exception NegativeValue: s needs to be positive.
00609         @exception NegativeValue: fs needs to be positive.
00610         @exception NegativeValue: Es needs to be positive.
00611         @exception NegativeValue: Ec needs to be positive if different from -1.
00612         @exception InconsistentGeometry: e should be smaller than half the depth and the width of the
section.
00613         """
00614         # Check
00615         if b < 0: raise NegativeValue()
00616         if L < 0: raise NegativeValue()
00617         if e < 0: raise NegativeValue()
00618         if fc > 0: raise PositiveValue()
00619         if D_bars < 0: raise NegativeValue()
00620         if n_bars < 0: raise NegativeValue()
00621         if fy < 0: raise NegativeValue()
00622         if Ey < 0: raise NegativeValue()
00623         if D_hoops < 0: raise NegativeValue()
00624         if s < 0: raise NegativeValue()
00625         if fs < 0: raise NegativeValue()
00626         if Es < 0: raise NegativeValue()
00627         if Ec != -1 and Ec < 0: raise NegativeValue()
00628         if e > b/2: raise InconsistentGeometry()
00629
00630         # Arguments
00631         self.bb = b
00632         self.LL = L
00633         self.ee = e
00634         self.fcfc = fc
00635         self.D_barsD_bars = D_bars
00636         self.n_barsn_bars = n_bars
00637         self.fyfy = fy
00638         self.EyEy = Ey
00639         self.D_hoopsD_hoops = D_hoops
00640         self.ss = s
00641         self.fsfs = fs
00642         self.EsEs = Es
00643         self.name_tagname_tag = name_tag
00644
00645         # Initialized the parameters that are dependent from others
00646         self.ReInitReInit(rho_s_vol, Ec)
00647
00648
00649     def ReInit(self, rho_s_vol = -1, Ec = -1):
00650         """
00651         Implementation of the homonym abstract method.
00652         See parent class DataManagement for detailed information.
00653
00654         @param rho_s_vol (float, optional): Ratio of the volume of transverse confining steel to the
volume of confined concrete core.
00655             Defaults to -1, e.g. computed according to Mander et Al. 1988.
00656         @param Ec (float): Young modulus for concrete. Defaults to -1, e.g. computed according to

```

```

Mander et Al. 1988.
00657     """
00658     # Precompute some members
00659     self.cl_hoopscl_hoops = self.ee + self.D_hoopsD_hoops/2.0 # centerline distance from the
border of the extreme confining hoops
00660     self.cl_bar scl_bar = self.ee + self.D_bar scl_bar/2.0 + self.D_hoopsD_hoops # centerline
distance from the border of the corner bars
00661     self.bcbc = self.bb - self.cl_hoopscl_hoops*2 # diameter of spiral (hoops) between bar
centerline
00662     self.AsAs = ComputeACircle(self.D_hoopsD_hoops)
00663
00664     # Arguments
00665     self.rho_s_volrho_s_vol = self.ComputeRhoVolComputeRhoVol() if rho_s_vol == -1 else rho_s_vol
00666     self.EcEc = self.ComputeEcComputeEc() if Ec == -1 else Ec
00667
00668     # Members
00669     self.AA = ComputeACircle(self.bb)
00670     self.AcAc = ComputeACircle(self.bcbc)
00671     self.AyAy = ComputeACircle(self.D_bar scl_bar)
00672     self.rho_bar scl_bar = ComputeRho(self.AyAy, self.n_bar scl_bar, self.AA)
00673     self.II = self.ComputeIComputeI()
00674
00675     # Data storage for loading/saving
00676     self.UpdateStoredDataUpdateStoredData()
00677
00678
00679     def UpdateStoredData(self):
00680         """
00681         Implementation of the homonym abstract method.
00682         See parent class DataManagement for detailed information.
00683         """
00684         self.data = [{"INFO_TYPE": "RCCircShape"}, # Tag for differentiating different data
00685                       [{"name_tag": self.name_tagname_tag,
00686                         ["b", self.bb],
00687                         ["bc", self.bcbc],
00688                         ["L", self.LL],
00689                         ["e", self.ee],
00690                         ["A", self.AA],
00691                         ["Ac", self.AcAc],
00692                         ["I", self.II],
00693                         ["fc", self.fcfc],
00694                         ["Ec", self.EcEc],
00695                         ["D_bar", self.D_bar scl_bar],
00696                         ["n_bar", self.n_bar scl_bar],
00697                         ["Ay", self.AyAy],
00698                         ["rho_bar", self.rho_bar scl_bar],
00699                         ["cl_bar", self.cl_bar scl_bar],
00700                         ["fy", self.fyfy],
00701                         ["Ey", self.EyEy],
00702                         ["D_hoops", self.D_hoopsD_hoops],
00703                         ["s", self.ss],
00704                         ["As", self.AsAs],
00705                         ["rho_s_vol", self.rho_s_volrho_s_vol],
00706                         ["cl_hoops", self.cl_hoopscl_hoops],
00707                         ["fs", self.fsfs],
00708                         ["Es", self.EsEs]]
00709
00710     def ShowInfo(self):
00711         """
00712         Implementation of the homonym abstract method.
00713         See parent class DataManagement for detailed information.
00714         """
00715         print("")
00716         print("Requested info for RC circular section of name tag = {}".format(self.name_tagname_tag))
00717         print("Width of the section b = {} mm".format(self.bb/mm_unit))
00718         print("Concrete cover e = {} mm".format(self.ee/mm_unit))
00719         print("Concrete area A = {} mm2".format(self.AA/mm2_unit))
00720         print("Core concrete area Ac = {} mm2".format(self.AcAc/mm2_unit))
00721         print("Unconfined concrete compressive strength fc = {} MPa".format(self.fcfc/MPa_unit))
00722         print("Young modulus for concrete Ec = {} GPa".format(self.EcEc/GPa_unit))
00723         print("Diameter of the reinforcing bars D_bar = {} mm and area of one bar Ay = {} mm2 with {}
bars".format(self.D_bar scl_bar/mm_unit, self.AyAy/mm2_unit, self.n_bar scl_bar))
00724         print("Diameter of the hoops D_hoops = {} mm and area of one stirrup As = {}
mm2".format(self.D_hoopsD_hoops/mm_unit, self.AsAs/mm2_unit))
00725         print("Ratio of area of longitudinal reinforcement to area of concrete section rho_bar = {}
".format(self.rho_bar scl_bar))
00726         print("Ratio of the volume of transverse confining steel to the volume of confined concrete
core rho_s = {}".format(self.rho_s_volrho_s_vol))
00727         print("Moment of inertia of the circular section I = {} mm4".format(self.II/mm4_unit))
00728         print("")
00729
00730
00731     def ComputeRhoVol(self):
00732         """
00733         Compute the ratio of the volume of transverse confining steel to the volume of confined
concrete core.
00734         (according to Mander et Al. 1988).

```

```

00735
00736     @returns float: Ratio.
00737     """
00738     vol_s = self.As*math.pi*self.bcbc
00739     vol_c = math.pi/4*self.bcbc**2*self.ss
00740
00741     return vol_s/vol_c
00742
00743
00744 def ComputeEc(self):
00745     """
00746     Compute Ec using the formula from Mander et Al. 1988.
00747
00748     @returns float: Young modulus of concrete.
00749     """
00750
00751     return 5000.0 * math.sqrt(-self.fcfc/MPa_unit) * MPa_unit
00752
00753
00754 def ComputeI(self):
00755     """
00756     Compute the moment of inertia of the circular section.
00757
00758     @returns float: Moment of inertia.
00759     """
00760     return self.bb**4*math.pi/64
00761
00762
00763 def ComputeACircle(D):
00764     """
00765     Function that computes the area of one circle (reinforcing bar or hoop).
00766
00767     @param D (float): Diameter of the circle (reinforcing bar or hoop).
00768
00769     @returns float: Area the circle (for reinforcing bars or hoops).
00770     """
00771     return D**2/4.0*math.pi
00772
00773
00774 def ComputeRho(A, nr, A_tot):
00775     """
00776     Compute the ratio of area of a reinforcement to area of a section.
00777
00778     @param A (float): Area of reinforcement.
00779     @param nr (float): Number of reinforcement (allow float for computing ratio with different area;
00780         just convert the other areas to one and compute the equivalent n).
00781     @param A_tot (float): Area of the concrete.
00782
00783     @returns float: Ratio.
00784     """
00785     return nr * A / A_tot

```

## 8.24 /media/carmine/DATA/Programmi/OpenSeesPyAssistant/Units.py File Reference

### Namespaces

- namespace [Units](#)

*Module with the units conversion and the definition of the units used as default (m, N, s).*

### Variables

- float [cm2\\_unit](#) = cm\_unit\*cm\_unit
- float [cm3\\_unit](#) = cm\_unit\*cm\_unit\*cm\_unit
- float [cm4\\_unit](#) = cm3\_unit\*cm\_unit
- float [cm\\_unit](#) = m\_unit\*1e-2
- float [dm2\\_unit](#) = dm\_unit\*dm\_unit
- float [dm3\\_unit](#) = dm\_unit\*dm\_unit\*dm\_unit
- float [dm4\\_unit](#) = dm3\_unit\*dm\_unit
- float [dm\\_unit](#) = m\_unit\*1e-1

- string `force_unit` = "N"
- float `ft2_unit` = `ft_unit*ft_unit`
- float `ft3_unit` = `ft_unit*ft_unit*ft_unit`
- float `ft4_unit` = `ft3_unit*ft_unit`
- float `ft_unit` = `m_unit*0.3048`
- float `GN_unit` = `N_unit*1e9`
- float `GPa_unit` = `Pa_unit*1e9`
- float `hours_unit` = `min_unit*60`
- float `inch2_unit` = `inch_unit*inch_unit`
- float `inch3_unit` = `inch_unit*inch_unit*inch_unit`
- float `inch4_unit` = `inch3_unit*inch_unit`
- float `inch_unit` = `m_unit*0.0254`
- float `kg_unit` = `N_unit*s_unit**2/m_unit`
- float `kip_unit` = `N_unit*4448.2216`
- float `km_unit` = `m_unit*1e3`
- float `kN_unit` = `N_unit*1e3`
- float `kNm_unit` = `kN_unit*m_unit`
- float `kNmm_unit` = `kN_unit*mm_unit`
- float `kPa_unit` = `Pa_unit*1e3`
- float `ksi_unit` = `psi_unit*1000`
- string `length_unit` = "m"
- float `m2_unit` = `m_unit*m_unit`
- float `m3_unit` = `m_unit*m_unit*m_unit`
- float `m4_unit` = `m3_unit*m_unit`
- float `m_unit` = 1.0
- float `mile_unit` = `m_unit*1609.34`
- float `min_unit` = `s_unit*60`
- float `mm2_unit` = `mm_unit*mm_unit`
- float `mm3_unit` = `mm_unit*mm_unit*mm_unit`
- float `mm4_unit` = `mm3_unit*mm_unit`
- float `mm_unit` = `m_unit*1e-3`
- float `MN_unit` = `N_unit*1e6`
- float `MNm_unit` = `MN_unit*m_unit`
- float `MNmm_unit` = `MN_unit*mm_unit`
- float `MPa_unit` = `Pa_unit*1e6`
- float `N_unit` = 1.0
- float `Nm_unit` = `N_unit*m_unit`
- float `Nmm_unit` = `N_unit*mm_unit`
- float `Pa_unit` = `N_unit/m2_unit`
- float `pound_unit` = `kg_unit*0.45359237`
- float `psi_unit` = `Pa_unit*6894.76`
- float `s_unit` = 1.0
- float `t_unit` = `kg_unit*1e3`
- string `time_unit` = "s"

## 8.25 Units.py

[Go to the documentation of this file.](#)

```
00001 """Module with the units conversion and the definition of the units used as default (m, N, s). \n
00002 Note that the decision of which unit for each measure (distance, force, mass, time) is equal to 1 is
      not arbitrary:
00003 for example the natural frequency is computed behind the scene by the OpenSeesPy framework, thus the
      stiffness of the structure divided by the mass should result in a unit of 1 (thus seconds). \n
00004 Furthermore, there are constants like the gravitational one g that is dependent on this decision. If
      the units are used in a consistent way (using this library), these issues can be avoided. \n
```



```

00005 Carmine Schipani, 2021
00006 """
00007
00008
00009 # Fundamental
00010 m_unit = 1.0
00011 length_unit = "m" # It's the length unit associated with 1 (fundamental)
00012 N_unit = 1.0
00013 force_unit = "N" # It's the force unit associated with 1 (fundamental)
00014 s_unit = 1.0
00015 time_unit = "s" # It's the time unit associated with 1 (fundamental)
00016
00017 # Distance
00018 mm_unit = m_unit*1e-3
00019 cm_unit = m_unit*1e-2
00020 dm_unit = m_unit*1e-1
00021 km_unit = m_unit*1e3
00022 inch_unit = m_unit*0.0254
00023 ft_unit = m_unit*0.3048
00024 mile_unit = m_unit*1609.34
00025
00026 # Area
00027 mm2_unit = mm_unit*mm_unit
00028 cm2_unit = cm_unit*cm_unit
00029 dm2_unit = dm_unit*dm_unit
00030 m2_unit = m_unit*m_unit
00031 inch2_unit = inch_unit*inch_unit
00032 ft2_unit = ft_unit*ft_unit
00033
00034 # Volume
00035 mm3_unit = mm_unit*mm_unit*mm_unit
00036 cm3_unit = cm_unit*cm_unit*cm_unit
00037 dm3_unit = dm_unit*dm_unit*dm_unit
00038 m3_unit = m_unit*m_unit*m_unit
00039 inch3_unit = inch_unit*inch_unit*inch_unit
00040 ft3_unit = ft_unit*ft_unit*ft_unit
00041
00042 # Moment of inertia
00043 mm4_unit = mm3_unit*mm_unit
00044 cm4_unit = cm3_unit*cm_unit
00045 dm4_unit = dm3_unit*dm_unit
00046 m4_unit = m3_unit*m_unit
00047 inch4_unit = inch3_unit*inch_unit
00048 ft4_unit = ft3_unit*ft_unit
00049
00050 # Force
00051 kN_unit = N_unit*1e3
00052 MN_unit = N_unit*1e6
00053 GN_unit = N_unit*1e9
00054 kip_unit = N_unit*4448.2216
00055
00056 # Moment (and rotational stiffnes (moment-rotation))
00057 Nm_unit = N_unit*m_unit
00058 kNm_unit = kN_unit*m_unit
00059 MNm_unit = MN_unit*m_unit
00060 Nmm_unit = N_unit*mm_unit
00061 kNmm_unit = kN_unit*mm_unit
00062 MNmm_unit = MN_unit*mm_unit
00063
00064 # Mass
00065 kg_unit = N_unit*s_unit**2/m_unit
00066 t_unit = kg_unit*1e3
00067 pound_unit = kg_unit*0.45359237
00068
00069 # Pressure/Stress
00070 Pa_unit = N_unit/m2_unit
00071 kPa_unit = Pa_unit*1e3
00072 MPa_unit = Pa_unit*1e6
00073 GPa_unit = Pa_unit*1e9
00074 psi_unit = Pa_unit*6894.76
00075 ksi_unit = psi_unit*1000
00076
00077 # Time
00078 min_unit = s_unit*60
00079 hours_unit = min_unit*60
00080

```



# Index

[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Analysis](#), 427  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Connections](#), 438  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Constants](#), 439, 440  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/DataManager](#), 440  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/ErrorHandler](#), 441, 442  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Fibers](#), 443  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Functions](#), 456  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Geometry](#), 462, 463  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Materials](#), 466, 468  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Members](#), 509, 511  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/README](#), 533  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Sections](#), 533  
[/media/carmine/DATA/Programmi/OpenSeesPyAssistant/Units](#), 543, 544  
[\\_\\_init\\_\\_](#)  
     Analysis, 68  
     ConfMander1988Circ, 85  
     ConfMander1988CircRCCircShape, 101  
     ConfMander1988Rect, 105  
     ConfMander1988RectRCRectShape, 125  
     ElasticElement, 131  
     ElasticElementSteelIIShape, 139  
     FibersCirc, 142  
     FibersCircRCCircShape, 151  
     FibersIIShape, 154  
     FibersIIShapeSteelIIShape, 162  
     FibersRect, 165  
     FibersRectRCRectShape, 175  
     ForceBasedElement, 178  
     ForceBasedElementFibersCircRCCircShape, 186  
     ForceBasedElementFibersIIShapeSteelIIShape, 189  
     ForceBasedElementFibersRectRCRectShape, 191  
     GIFBEelement, 195  
     GIFBEelementFibersCircRCCircShape, 206  
     GIFBEelementFibersRectRCRectShape, 209  
     GIFBEelementRCCircShape, 211  
     GIFBEelementRCCRectShape, 214  
     GMP1970, 219  
     GMP1970RCRectShape, 226  
     Gupta1999, 230  
     Gupta1999SteelIIShape, 241  
     IDGenerator, 244  
     IMKSteelIIShape, 254  
     ModifiedIMKSteelIIShape, 274  
     PanelZone, 278  
     PanelZoneRCS, 287  
     PanelZoneSteelIIShape, 290  
     PanelZoneSteelIIShapeGupta1999, 292  
     PanelZoneSteelIIShapeSkiadopoulos2021, 295  
     RCCircShape, 298  
     RCRectShape, 309  
     RCSquareShape, 323  
     Skiadopoulos2021, 328  
     Skiadopoulos2021RCS, 345  
     Skiadopoulos2021SteelIIShape, 348  
     SpringBasedElement, 351  
     SpringBasedElementModifiedIMKSteelIIShape, 360  
     SpringBasedElementSteelIIShape, 364  
     SteelIIShape, 367  
     UnconfMander1988, 379  
     UnconfMander1988RCCircShape, 389  
     UnconfMander1988RCRectShape, 392  
     UniaxialBilinear, 394  
     UniaxialBilinearSteelIIShape, 401  
     UVC, 404  
     UVC calibrated, 411  
     UVC calibratedRCCircShape, 414  
     UVC calibratedRCRectShape, 416  
     UVC calibratedSteelIIShapeFlange, 418  
     UVC calibratedSteelIIShapeWeb, 420  
     WrongNodeIDConvention, 423  
     ZeroLength, 425  
  
     A  
         ElasticElement, 136  
         RCCircShape, 303  
         RCRectShape, 317  
         SpringBasedElement, 356  
         SteelIIShape, 374  
  
     a  
         ModifiedIMK, 267  
         UVC, 408  
  
     a1  
         GMP1970, 223  
  
     a2

- GMP1970, [223](#)
- a3
  - GMP1970, [223](#)
- a4
  - GMP1970, [223](#)
- A\_rigid
  - PanelZone, [283](#)
- a\_s
  - Gupta1999, [235](#)
  - ModifiedIMK, [267](#)
  - Skiadopoulos2021, [335](#)
- Ac
  - ConfMander1988Circ, [95](#)
  - ConfMander1988Rect, [118](#)
  - RCCircShape, [304](#)
  - RCRectShape, [317](#)
- Acc
  - ConfMander1988Circ, [95](#)
  - ConfMander1988Rect, [118](#)
- Ae
  - ConfMander1988Circ, [96](#)
  - ConfMander1988Rect, [118](#)
- Ai
  - ConfMander1988Rect, [118](#)
- algo
  - Analysis, [82](#)
- allow\_smaller\_step
  - Analysis, [82](#)
- alpha\_i
  - FibersCirc, [146](#)
- Analysis, [67](#)
  - \_\_init\_\_, [68](#)
  - algo, [82](#)
  - allow\_smaller\_step, [82](#)
  - data\_dir, [82](#)
  - DeformedShape, [70](#)
  - FiberResponse, [71](#)
  - Gravity, [72](#)
  - LateralForce, [74](#)
  - load\_case, [82](#)
  - LoadingProtocol, [77](#)
  - max\_iter, [83](#)
  - name\_ODB, [83](#)
  - Pushover, [79](#)
  - test\_opt, [83](#)
  - test\_type, [83](#)
  - tol, [83](#)
- AnalysisAndPostProcessing, [13](#)
- array\_fl2
  - ConfMander1988Rect, [118](#)
- As
  - RCCircShape, [304](#)
  - RCRectShape, [317](#)
- Ay
  - FibersCirc, [147](#)
  - FibersRect, [171](#)
  - RCCircShape, [304](#)
  - RCRectShape, [318](#)
- b
  - FibersCirc, [147](#)
  - FibersRect, [171](#)
  - GMP1970, [223](#)
  - RCCircShape, [304](#)
  - RCRectShape, [318](#)
  - UniaxialBilinear, [399](#)
  - UVC, [408](#)
- bars\_mat\_ID
  - FibersCirc, [147](#)
  - FibersRect, [171](#)
- bars\_position\_x
  - RCRectShape, [318](#)
- bars\_ranges\_position\_y
  - RCRectShape, [318](#)
- bars\_x
  - FibersRect, [171](#)
- bc
  - ConfMander1988Circ, [96](#)
  - ConfMander1988Rect, [119](#)
  - RCCircShape, [304](#)
  - RCRectShape, [318](#)
- beam
  - Gupta1999SteelShape, [243](#)
  - PanelZoneRCS, [288](#)
  - PanelZoneSteelShape, [291](#)
  - PanelZoneSteelShapeGupta1999, [293](#)
  - PanelZoneSteelShapeSkiadopoulos2021, [296](#)
  - Skiadopoulos2021RCS, [346](#)
  - Skiadopoulos2021SteelShape, [349](#)
- beam\_section\_name\_tag
  - Gupta1999, [235](#)
  - Gupta1999SteelShape, [243](#)
  - PanelZone, [283](#)
  - PanelZoneRCS, [288](#)
  - PanelZoneSteelShape, [291](#)
  - Skiadopoulos2021, [335](#)
  - Skiadopoulos2021RCS, [346](#)
  - Skiadopoulos2021SteelShape, [349](#)
- beta
  - ConfMander1988Circ, [96](#)
  - ConfMander1988Rect, [119](#)
  - Gupta1999, [235](#)
  - Skiadopoulos2021, [335](#)
  - UnconfMander1988, [386](#)
- bf
  - ModifiedIMK, [267](#)
  - SteelShape, [374](#)
- bf\_b
  - FibersIShape, [158](#)
- bf\_c
  - Gupta1999, [235](#)
  - Skiadopoulos2021, [335](#)
- bf\_t
  - FibersIShape, [158](#)
- Bilin
  - ModifiedIMK, [258](#)
- bottom\_flange\_mat\_ID

- FibersIShape, 158
- calibration
  - UVCCalibrated, 413
- Cf1
  - Skiadopoulos2021, 335
- Cf1\_tests
  - Skiadopoulos2021, 336
- Cf4
  - Skiadopoulos2021, 336
- Cf4\_tests
  - Skiadopoulos2021, 336
- Cf6
  - Skiadopoulos2021, 336
- Cf6\_tests
  - Skiadopoulos2021, 336
- CheckApplicability
  - ConfMander1988Circ, 88
  - ConfMander1988Rect, 109
  - GMP1970, 220
  - Gupta1999, 232
  - MaterialModels, 249
  - ModifiedIMK, 258
  - Skiadopoulos2021, 331
  - UnconfMander1988, 381
  - UniaxialBilinear, 396
  - UVC, 405
- cK
  - UVC, 408
- cl\_bars
  - RCCircShape, 304
  - RCCRectShape, 318
- cl\_hoops
  - RCCircShape, 305
  - RCCRectShape, 319
- cm2\_unit
  - Units, 57
- cm3\_unit
  - Units, 57
- cm4\_unit
  - Units, 58
- cm\_unit
  - Units, 58
- col
  - Gupta1999SteelIShape, 243
  - PanelZoneRCS, 288
  - PanelZoneSteelIShape, 291
  - PanelZoneSteelIShapeGupta1999, 293
  - PanelZoneSteelIShapeSkiadopoulos2021, 296
  - Skiadopoulos2021SteelIShape, 349
- col\_section\_name\_tag
  - Gupta1999, 236
  - Gupta1999SteelIShape, 243
  - PanelZone, 284
  - PanelZoneRCS, 288
  - PanelZoneSteelIShape, 291
  - Skiadopoulos2021, 336
  - Skiadopoulos2021SteelIShape, 349
- Compute\_ec
  - ConfMander1988Circ, 88
  - ConfMander1988Rect, 109
  - UnconfMander1988, 381
- Compute\_ecc
  - ConfMander1988Circ, 89
  - ConfMander1988Rect, 109
- Compute\_eccu
  - ConfMander1988Circ, 89
  - ConfMander1988Rect, 110
- Compute\_ecp
  - ConfMander1988Circ, 90
  - ConfMander1988Rect, 110
  - UnconfMander1988, 381
- Compute\_ecu
  - ConfMander1988Circ, 90
  - ConfMander1988Rect, 111
  - UnconfMander1988, 382
- Compute\_et
  - ConfMander1988Circ, 91
  - ConfMander1988Rect, 111
  - UnconfMander1988, 382
- Compute\_fct
  - ConfMander1988Circ, 91
  - ConfMander1988Rect, 112
  - UnconfMander1988, 383
- Compute\_iy
  - SteelIShape, 369
- Compute\_iz
  - SteelIShape, 369
- ComputeA
  - RCCRectShape, 313
  - SteelIShape, 369
- Computea
  - ModifiedIMK, 259
- Computea\_s
  - ModifiedIMK, 259
- ComputeAc
  - RCCRectShape, 313
- ComputeACircle
  - Section, 55
- ComputeAi
  - ConfMander1988Rect, 112
- ComputeConfinementFactor
  - ConfMander1988Rect, 113
- ComputeEc
  - RCCircShape, 300
  - RCCRectShape, 313
- Computel
  - RCCircShape, 301
- Computelp
  - GIFBElement, 197
- Computely
  - RCCRectShape, 314
  - SteelIShape, 370
- Computelz
  - RCCRectShape, 314
  - SteelIShape, 370
- ComputeK

- ModifiedIMK, 259
- ComputeKe
  - ModifiedIMK, 260
- ComputeLp
  - GIFBElement, 198
- ComputeMc
  - ModifiedIMK, 260
- ComputeMyStar
  - ModifiedIMK, 261
- ComputeNrBars
  - RCCRectShape, 314
- ComputeRefEnergyDissipationCap
  - ModifiedIMK, 261
- ComputeRho
  - Section, 55
- ComputeRhoVol
  - RCCircShape, 301
- ComputeTheta\_p
  - ModifiedIMK, 262
- ComputeTheta\_pc
  - ModifiedIMK, 263
- ComputeTheta\_u
  - ModifiedIMK, 263
- ComputeTheta\_y
  - ModifiedIMK, 264
- ComputeWply
  - SteelShape, 371
- ComputeWplz
  - SteelShape, 371
- Concrete01
  - ConfMander1988Circ, 92
  - ConfMander1988Rect, 114
  - UnconfMander1988, 383
- Concrete01Funct
  - MaterialModels, 45
- Concrete04
  - ConfMander1988Circ, 92
  - ConfMander1988Rect, 115
  - UnconfMander1988, 384
- Concrete04Funct
  - MaterialModels, 47
- conf\_mat\_ID
  - FibersCirc, 147
  - FibersRect, 171
- ConfMander1988Circ, 84
  - \_\_init\_\_, 85
  - Ac, 95
  - Acc, 95
  - Ae, 96
  - bc, 96
  - beta, 96
  - CheckApplicability, 88
  - Compute\_ec, 88
  - Compute\_ecc, 89
  - Compute\_eccu, 89
  - Compute\_ecp, 90
  - Compute\_ecu, 90
  - Compute\_et, 91
  - Compute\_fct, 91
  - Concrete01, 92
  - Concrete04, 92
  - D\_bars, 96
  - D\_hoops, 96
  - data, 96
  - Ec, 97
  - ec, 97
  - ecc, 97
  - eccu, 97
  - ecp, 97
  - ecu, 97
  - esu, 98
  - et, 98
  - fc, 98
  - fcc, 98
  - fct, 98
  - fl, 98
  - fl\_prime, 99
  - fs, 99
  - ID, 99
  - Initialized, 99
  - K\_combo, 99
  - ke, 99
  - nr\_bars, 100
  - Relnit, 93
  - rho\_cc, 100
  - rho\_s\_vol, 100
  - s, 100
  - section\_name\_tag, 100
  - ShowInfo, 94
  - UpdateStoredData, 95
- ConfMander1988CircRCCircShape, 101
  - \_\_init\_\_, 101
  - section, 103
  - section\_name\_tag, 103
- ConfMander1988Rect, 103
  - \_\_init\_\_, 105
  - Ac, 118
  - Acc, 118
  - Ae, 118
  - Ai, 118
  - array\_fl2, 118
  - bc, 119
  - beta, 119
  - CheckApplicability, 109
  - Compute\_ec, 109
  - Compute\_ecc, 109
  - Compute\_eccu, 110
  - Compute\_ecp, 110
  - Compute\_ecu, 111
  - Compute\_et, 111
  - Compute\_fct, 112
  - ComputeAi, 112
  - ComputeConfinementFactor, 113
  - Concrete01, 114
  - Concrete04, 115
  - curve\_fl1, 119

- D\_bars, [119](#)
- D\_hoops, [119](#)
- data, [119](#)
- dc, [120](#)
- Ec, [120](#)
- ec, [120](#)
- ecc, [120](#)
- eccu, [120](#)
- ecp, [120](#)
- ecu, [121](#)
- esu, [121](#)
- et, [121](#)
- fc, [121](#)
- fcc, [121](#)
- fct, [121](#)
- fl\_x, [122](#)
- fl\_y, [122](#)
- fs, [122](#)
- ID, [122](#)
- Initialized, [122](#)
- K\_combo, [122](#)
- ke, [123](#)
- nr\_bars, [123](#)
- Relnit, [115](#)
- rho\_cc, [123](#)
- rho\_s\_x, [123](#)
- rho\_s\_y, [123](#)
- s, [123](#)
- section\_name\_tag, [124](#)
- ShowInfo, [116](#)
- UpdateStoredData, [117](#)
- wx\_bottom, [124](#)
- wx\_top, [124](#)
- wy, [124](#)
- ConfMander1988RectRCRectShape, [125](#)
  - \_\_init\_\_, [125](#)
  - section, [127](#)
  - section\_name\_tag, [127](#)
- Connections, [13](#)
  - Pin, [14](#)
  - RigidSupport, [14](#)
  - RotationalSpring, [15](#)
- Constants, [16](#)
  - G\_CONST, [17](#)
  - MAX\_ITER, [17](#)
  - MAX\_ITER\_INTEGRATION, [17](#)
  - RIGID, [17](#)
  - TOL, [17](#)
  - TOL\_INTEGRATION, [17](#)
  - ZERO, [18](#)
- cR1
  - GMP1970, [223](#)
- cR2
  - GMP1970, [224](#)
- create\_fiber\_section
  - Fibers, [20](#)
- CreateFibers
  - FibersCirc, [144](#)
  - FibersIShape, [156](#)
  - FibersRect, [168](#)
- CreateMember
  - ElasticElement, [133](#)
  - ForceBasedElement, [180](#)
  - GIFBElement, [198](#)
  - PanelZone, [280](#)
  - SpringBasedElement, [353](#)
- current\_element\_ID
  - IDGenerator, [246](#)
- current\_fiber\_ID
  - IDGenerator, [246](#)
- current\_mat\_ID
  - IDGenerator, [247](#)
- current\_node\_ID
  - IDGenerator, [247](#)
- curve\_fl1
  - ConfMander1988Rect, [119](#)
- Cw1
  - Skiadopoulos2021, [337](#)
- Cw1\_tests
  - Skiadopoulos2021, [337](#)
- Cw4
  - Skiadopoulos2021, [337](#)
- Cw4\_tests
  - Skiadopoulos2021, [337](#)
- Cw6
  - Skiadopoulos2021, [337](#)
- Cw6\_tests
  - Skiadopoulos2021, [337](#)
- d
  - FibersIShape, [159](#)
  - FibersRect, [171](#)
  - ModifiedIMK, [267](#)
  - RCRectShape, [319](#)
  - SteelIShape, [374](#)
- d\_b
  - Gupta1999, [236](#)
  - Skiadopoulos2021, [338](#)
- D\_bars
  - ConfMander1988Circ, [96](#)
  - ConfMander1988Rect, [119](#)
  - FibersCirc, [147](#)
  - GIFBElement, [202](#)
  - RCCircShape, [305](#)
  - RCRectShape, [319](#)
- d\_c
  - Gupta1999, [236](#)
  - Skiadopoulos2021, [338](#)
- D\_hoops
  - ConfMander1988Circ, [96](#)
  - ConfMander1988Rect, [119](#)
  - FibersCirc, [147](#)
  - FibersRect, [172](#)
  - RCCircShape, [305](#)
  - RCRectShape, [319](#)
- data
  - ConfMander1988Circ, [96](#)

- ConfMander1988Rect, [119](#)
- ElasticElement, [136](#)
- FibersCirc, [148](#)
- FibersIShape, [159](#)
- FibersRect, [172](#)
- ForceBasedElement, [183](#)
- GIFBElement, [202](#)
- GMP1970, [224](#)
- Gupta1999, [236](#)
- ModifiedIMK, [268](#)
- PanelZone, [284](#)
- RCCircShape, [305](#)
- RCRectShape, [319](#)
- Skiadopoulos2021, [338](#)
- SpringBasedElement, [357](#)
- SteelIShape, [374](#)
- UnconfMander1988, [387](#)
- UniaxialBilinear, [399](#)
- UVC, [408](#)
- data\_dir
  - Analysis, [82](#)
- DataManagement, [18](#), [127](#)
  - Relnit, [128](#)
  - SaveData, [128](#)
  - ShowInfo, [129](#)
  - UpdateStoredData, [130](#)
- dc
  - ConfMander1988Rect, [120](#)
  - RCRectShape, [319](#)
- DefineFrameNodes
  - GeometryTemplate, [37](#)
- DefineFrameNodesAndElementsSteelIShape
  - GeometryTemplate, [38](#)
- DefinePanelZoneElements
  - MemberModel, [51](#)
- DefinePanelZoneNodes
  - MemberModel, [53](#)
- DefineSubassemblageNodes
  - GeometryTemplate, [41](#)
- DeformedShape
  - Analysis, [70](#)
- DInf
  - UVC, [409](#)
- discr\_bottom\_flange
  - FibersIShape, [159](#)
- discr\_core
  - FibersCirc, [148](#)
  - FibersRect, [172](#)
- discr\_cover
  - FibersCirc, [148](#)
- discr\_cover\_lateral
  - FibersRect, [172](#)
- discr\_cover\_topbottom
  - FibersRect, [172](#)
- discr\_top\_flange
  - FibersIShape, [159](#)
- discr\_web
  - FibersIShape, [159](#)
- DiscretizeLinearly
  - FunctionalFeatures, [25](#)
- DiscretizeLoadProtocol
  - FunctionalFeatures, [27](#)
- dm2\_unit
  - Units, [58](#)
- dm3\_unit
  - Units, [58](#)
- dm4\_unit
  - Units, [58](#)
- dm\_unit
  - Units, [58](#)
- dmg1
  - Gupta1999, [236](#)
  - Skiadopoulos2021, [338](#)
- dmg2
  - Gupta1999, [236](#)
  - Skiadopoulos2021, [338](#)
- E
  - ElasticElement, [136](#)
  - Gupta1999, [237](#)
  - ModifiedIMK, [268](#)
  - PanelZone, [284](#)
  - Skiadopoulos2021, [338](#)
  - SpringBasedElement, [357](#)
  - SteelIShape, [374](#)
- e
  - FibersCirc, [148](#)
  - FibersRect, [172](#)
  - RCCircShape, [305](#)
  - RCRectShape, [320](#)
- Ec
  - ConfMander1988Circ, [97](#)
  - ConfMander1988Rect, [120](#)
  - RCCircShape, [305](#)
  - RCRectShape, [320](#)
  - UnconfMander1988, [387](#)
- ec
  - ConfMander1988Circ, [97](#)
  - ConfMander1988Rect, [120](#)
  - UnconfMander1988, [387](#)
- ecc
  - ConfMander1988Circ, [97](#)
  - ConfMander1988Rect, [120](#)
- eccu
  - ConfMander1988Circ, [97](#)
  - ConfMander1988Rect, [120](#)
- ecp
  - ConfMander1988Circ, [97](#)
  - ConfMander1988Rect, [120](#)
  - UnconfMander1988, [387](#)
- ecu
  - ConfMander1988Circ, [97](#)
  - ConfMander1988Rect, [121](#)
  - UnconfMander1988, [387](#)
- ElasticElement, [130](#)
  - \_\_init\_\_, [131](#)
  - A, [136](#)



- CreateMember, 133
- data, 136
- E, 136
- element\_array, 136
- element\_ID, 137
- geo\_transf\_ID, 137
- Initialized, 137
- iNode\_ID, 137
- ly, 137
- jNode\_ID, 137
- Record, 133
- RecordNodeDef, 134
- Relnit, 134
- section\_name\_tag, 138
- ShowInfo, 135
- UpdateStoredData, 135
- ElasticElementSteelIShape, 138
  - \_\_init\_\_, 139
  - section, 139
  - section\_name\_tag, 140
- ele\_orientation
  - SpringBasedElement, 357
- element
  - ZeroLength, 425
- element\_array
  - ElasticElement, 136
  - ForceBasedElement, 183
  - GIFBElement, 202
  - PanelZone, 284
  - SpringBasedElement, 357
- element\_ID
  - ElasticElement, 137
  - ForceBasedElement, 183
  - GIFBElement, 202
  - SpringBasedElement, 357
- ErrorHandling, 18
- Es
  - RCCircShape, 306
  - RCCRectShape, 320
- esu
  - ConfMander1988Circ, 98
  - ConfMander1988Rect, 121
- et
  - ConfMander1988Circ, 98
  - ConfMander1988Rect, 121
  - UnconfMander1988, 387
- Ey
  - GMP1970, 224
  - RCCircShape, 306
  - RCCRectShape, 320
  - UniaxialBilinear, 399
  - UVC, 409
- ey
  - UniaxialBilinear, 399
- fc
  - ConfMander1988Circ, 98
  - ConfMander1988Rect, 121
  - RCCircShape, 306
  - RCCRectShape, 320
  - UnconfMander1988, 388
- fcc
  - ConfMander1988Circ, 98
  - ConfMander1988Rect, 121
- fct
  - ConfMander1988Circ, 98
  - ConfMander1988Rect, 121
  - UnconfMander1988, 388
- fib\_sec
  - FibersCirc, 148
  - FibersIShape, 159
  - FibersRect, 173
- fiber\_ID
  - ForceBasedElement, 183
  - GIFBElement, 202
- FiberResponse
  - Analysis, 71
- Fibers, 19, 140
  - create\_fiber\_section, 20
  - plot\_fiber\_section, 21
- FibersCirc, 141
  - \_\_init\_\_, 142
  - alpha\_i, 146
  - Ay, 147
  - b, 147
  - bars\_mat\_ID, 147
  - conf\_mat\_ID, 147
  - CreateFibers, 144
  - D\_bars, 147
  - D\_hoops, 147
  - data, 148
  - discr\_core, 148
  - discr\_cover, 148
  - e, 148
  - fib\_sec, 148
  - GJ, 148
  - ID, 149
  - Initialized, 149
  - n\_bars, 149
  - r\_bars, 149
  - r\_core, 149
  - Relnit, 144
  - section\_name\_tag, 149
  - ShowInfo, 145
  - unconf\_mat\_ID, 150
  - UpdateStoredData, 146
- FibersCircRCCircShape, 150
  - \_\_init\_\_, 151
  - section, 152
  - section\_name\_tag, 152
- FibersIShape, 152
  - \_\_init\_\_, 154
  - bf\_b, 158
  - bf\_t, 158
  - bottom\_flange\_mat\_ID, 158
  - CreateFibers, 156
  - d, 159

- data, 159
- discr\_bottom\_flange, 159
- discr\_top\_flange, 159
- discr\_web, 159
- fib\_sec, 159
- GJ, 160
- ID, 160
- Initialized, 160
- Relnit, 156
- section\_name\_tag, 160
- ShowInfo, 157
- tf\_b, 160
- tf\_t, 160
- top\_flange\_mat\_ID, 161
- tw, 161
- UpdateStoredData, 158
- web\_mat\_ID, 161
- FibersIShapeSteelShape, 161
  - \_\_init\_\_, 162
  - section, 163
  - section\_name\_tag, 163
- FibersRect, 164
  - \_\_init\_\_, 165
  - Ay, 171
  - b, 171
  - bars\_mat\_ID, 171
  - bars\_x, 171
  - conf\_mat\_ID, 171
  - CreateFibers, 168
  - d, 171
  - D\_hoops, 172
  - data, 172
  - discr\_core, 172
  - discr\_cover\_lateral, 172
  - discr\_cover\_topbottom, 172
  - e, 172
  - fib\_sec, 173
  - GJ, 173
  - ID, 173
  - Initialized, 173
  - ranges\_y, 173
  - rebarYZ, 173
  - Relnit, 168
  - section\_name\_tag, 174
  - ShowInfo, 169
  - unconf\_mat\_ID, 174
  - UpdateStoredData, 170
- FibersRectRCRectShape, 174
  - \_\_init\_\_, 175
  - section, 176
  - section\_name\_tag, 176
- fl
  - ConfMander1988Circ, 98
- fl\_prime
  - ConfMander1988Circ, 99
- fl\_x
  - ConfMander1988Rect, 122
- fl\_y
  - ConfMander1988Rect, 122
- force\_unit
  - Units, 59
- ForceBasedElement, 177
  - \_\_init\_\_, 178
  - CreateMember, 180
  - data, 183
  - element\_array, 183
  - element\_ID, 183
  - fiber\_ID, 183
  - geo\_transf\_ID, 183
  - Initialized, 184
  - iNode\_ID, 184
  - integration\_type, 184
  - Ip, 184
  - jNode\_ID, 184
  - max\_iter, 184
  - new\_integration\_ID, 185
  - Record, 180
  - RecordNodeDef, 180
  - Relnit, 181
  - section\_name\_tag, 185
  - ShowInfo, 182
  - tol, 185
  - UpdateStoredData, 182
- ForceBasedElementFibersCircRCCircShape, 185
  - \_\_init\_\_, 186
  - section, 187
  - section\_name\_tag, 187
- ForceBasedElementFibersIShapeSteelShape, 188
  - \_\_init\_\_, 189
  - section, 190
  - section\_name\_tag, 190
- ForceBasedElementFibersRectRCRectShape, 191
  - \_\_init\_\_, 191
  - section, 193
  - section\_name\_tag, 193
- fs
  - ConfMander1988Circ, 99
  - ConfMander1988Rect, 122
  - RCCircShape, 306
  - RCRectShape, 320
- ft2\_unit
  - Units, 59
- ft3\_unit
  - Units, 59
- ft4\_unit
  - Units, 59
- ft\_unit
  - Units, 59
- FunctionalFeatures, 24
  - DiscretizeLinearly, 25
  - DiscretizeLoadProtocol, 27
  - GridIDConvention, 28
  - IDConvention, 30
  - NodesOrientation, 31
  - OffsetNodeIDConvention, 32
  - plot\_member, 33

- plot\_nodes, [34](#)
  - ProgressingPercentage, [35](#)
- Fy
  - Gupta1999, [237](#)
  - ModifiedIMK, [268](#)
  - Skiadopoulos2021, [339](#)
  - SteelShape, [374](#)
- fy
  - GIFBElement, [202](#)
  - GMP1970, [224](#)
  - RCCircShape, [306](#)
  - RCRectShape, [321](#)
  - UniaxialBilinear, [400](#)
  - UVC, [409](#)
- Fy\_web
  - SteelShape, [375](#)
- G
  - Gupta1999, [237](#)
  - Skiadopoulos2021, [339](#)
- G\_CONST
  - Constants, [17](#)
- gamma1\_y
  - Gupta1999, [237](#)
- gamma2\_y
  - Gupta1999, [237](#)
- gamma3\_y
  - Gupta1999, [237](#)
- Gamma\_1
  - Skiadopoulos2021, [339](#)
- Gamma\_4
  - Skiadopoulos2021, [339](#)
- Gamma\_6
  - Skiadopoulos2021, [339](#)
- gamma\_rm
  - ModifiedIMK, [268](#)
- gammaK
  - UVC, [409](#)
- GenerateIDElement
  - IDGenerator, [245](#)
- GenerateIDFiber
  - IDGenerator, [245](#)
- GenerateIDMat
  - IDGenerator, [245](#)
- GenerateIDNode
  - IDGenerator, [246](#)
- geo\_transf\_ID
  - ElasticElement, [137](#)
  - ForceBasedElement, [183](#)
  - GIFBElement, [203](#)
  - PanelZone, [284](#)
  - SpringBasedElement, [357](#)
- GeometryTemplate, [36](#)
  - DefineFrameNodes, [37](#)
  - DefineFrameNodesAndElementsSteelShape, [38](#)
  - DefineSubassemblageNodes, [41](#)
  - Initialize2DModel, [43](#)
- GIFBElement, [193](#)
  - \_\_init\_\_, [195](#)
- Computelp, [197](#)
- ComputeLp, [198](#)
- CreateMember, [198](#)
- D\_bars, [202](#)
- data, [202](#)
- element\_array, [202](#)
- element\_ID, [202](#)
- fiber\_ID, [202](#)
- fy, [202](#)
- geo\_transf\_ID, [203](#)
- Initialized, [203](#)
- iNode\_ID, [203](#)
- lp, [203](#)
- jNode\_ID, [203](#)
- L, [203](#)
- lambda\_i, [204](#)
- lambda\_j, [204](#)
- Lp, [204](#)
- max\_iter, [204](#)
- max\_tol, [204](#)
- min\_tol, [204](#)
- new\_integration\_ID, [205](#)
- Record, [198](#)
- RecordNodeDef, [199](#)
- Relnit, [199](#)
- section\_name\_tag, [205](#)
- ShowInfo, [200](#)
- UpdateStoredData, [201](#)
- GIFBElementFibersCircRCCircShape, [205](#)
  - \_\_init\_\_, [206](#)
  - section, [207](#)
  - section\_name\_tag, [208](#)
- GIFBElementFibersRectRCRectShape, [208](#)
  - \_\_init\_\_, [209](#)
  - section, [210](#)
  - section\_name\_tag, [210](#)
- GIFBElementRCCircShape, [211](#)
  - \_\_init\_\_, [211](#)
  - section, [213](#)
  - section\_name\_tag, [213](#)
- GIFBElementRCRectShape, [214](#)
  - \_\_init\_\_, [214](#)
  - section, [216](#)
  - section\_name\_tag, [216](#)
- GJ
  - FibersCirc, [148](#)
  - FibersIShape, [160](#)
  - FibersRect, [173](#)
- GMP1970, [217](#)
  - \_\_init\_\_, [219](#)
  - a1, [223](#)
  - a2, [223](#)
  - a3, [223](#)
  - a4, [223](#)
  - b, [223](#)
  - CheckApplicability, [220](#)
  - cR1, [223](#)
  - cR2, [224](#)

- data, 224
- Ey, 224
- fy, 224
- ID, 224
- Initialized, 224
- R0, 225
- Relnit, 221
- section\_name\_tag, 225
- ShowInfo, 221
- Steel02, 222
- UpdateStoredData, 222
- GMP1970RCRectShape, 225
  - \_\_init\_\_, 226
  - section, 227
  - section\_name\_tag, 227
- GN\_unit
  - Units, 59
- GPa\_unit
  - Units, 60
- Gravity
  - Analysis, 72
- GridIDConvention
  - FunctionalFeatures, 28
- Gupta1999, 228
  - \_\_init\_\_, 230
  - a\_s, 235
  - beam\_section\_name\_tag, 235
  - beta, 235
  - bf\_c, 235
  - CheckApplicability, 232
  - col\_section\_name\_tag, 236
  - d\_b, 236
  - d\_c, 236
  - data, 236
  - dmg1, 236
  - dmg2, 236
  - E, 237
  - Fy, 237
  - G, 237
  - gamma1\_y, 237
  - gamma2\_y, 237
  - gamma3\_y, 237
  - Hysteretic, 232
  - I\_c, 238
  - ID, 238
  - Initialized, 238
  - Ke, 238
  - Kp, 238
  - M1y, 238
  - M2y, 239
  - M3y, 239
  - pinchx, 239
  - pinchy, 239
  - Relnit, 233
  - Ry, 239
  - ShowInfo, 233
  - t\_dp, 239
  - t\_p, 240
  - t\_pz, 240
  - tf\_b, 240
  - tf\_c, 240
  - UpdateStoredData, 234
  - Vy, 240
- Gupta1999SteelShape, 241
  - \_\_init\_\_, 241
  - beam, 243
  - beam\_section\_name\_tag, 243
  - col, 243
  - col\_section\_name\_tag, 243
- h\_1
  - ModifiedIMK, 268
  - SteelShape, 375
- hours\_unit
  - Units, 60
- Hysteretic
  - Gupta1999, 232
  - Skiadopoulos2021, 332
- I
  - RCCircShape, 306
- I\_c
  - Gupta1999, 238
  - Skiadopoulos2021, 339
- I\_rigid
  - PanelZone, 284
- ID
  - ConfMander1988Circ, 99
  - ConfMander1988Rect, 122
  - FibersCirc, 149
  - FibersIShape, 160
  - FibersRect, 173
  - GMP1970, 224
  - Gupta1999, 238
  - ModifiedIMK, 268
  - Skiadopoulos2021, 340
  - UnconfMander1988, 388
  - UniaxialBilinear, 400
  - UVC, 409
- IDConvention
  - FunctionalFeatures, 30
- IDGenerator, 244
  - \_\_init\_\_, 244
  - current\_element\_ID, 246
  - current\_fiber\_ID, 246
  - current\_mat\_ID, 247
  - current\_node\_ID, 247
  - GenerateIDElement, 245
  - GenerateIDFiber, 245
  - GenerateIDMat, 245
  - GenerateIDNode, 246
- inch2\_unit
  - Units, 60
- inch3\_unit
  - Units, 60
- inch4\_unit
  - Units, 60

- inch\_unit
  - Units, [60](#)
- InconsistentGeometry, [247](#)
- Initialize2DModel
  - GeometryTemplate, [43](#)
- Initialized
  - ConfMander1988Circ, [99](#)
  - ConfMander1988Rect, [122](#)
  - ElasticElement, [137](#)
  - FibersCirc, [149](#)
  - FibersIShape, [160](#)
  - FibersRect, [173](#)
  - ForceBasedElement, [184](#)
  - GIFBElement, [203](#)
  - GMP1970, [224](#)
  - Gupta1999, [238](#)
  - ModifiedIMK, [269](#)
  - PanelZone, [285](#)
  - Skiadopoulos2021, [340](#)
  - SpringBasedElement, [358](#)
  - UnconfMander1988, [388](#)
  - UniaxialBilinear, [400](#)
  - UVC, [409](#)
- iNode\_ID
  - ElasticElement, [137](#)
  - ForceBasedElement, [184](#)
  - GIFBElement, [203](#)
  - PanelZone, [285](#)
  - SpringBasedElement, [358](#)
- iNode\_ID\_spring
  - SpringBasedElement, [358](#)
- integration\_type
  - ForceBasedElement, [184](#)
- Ip
  - ForceBasedElement, [184](#)
  - GIFBElement, [203](#)
- iSpring\_ID
  - SpringBasedElement, [358](#)
- Iy
  - ElasticElement, [137](#)
  - RCCRectShape, [321](#)
  - SteelIShape, [375](#)
- Iy
  - SteelIShape, [375](#)
- Iy\_mod
  - ModifiedIMK, [269](#)
  - SpringBasedElement, [358](#)
  - SteelIShape, [375](#)
- Iz
  - RCCRectShape, [321](#)
  - SteelIShape, [375](#)
- iz
  - ModifiedIMK, [269](#)
  - SteelIShape, [376](#)
- jNode\_ID
  - ElasticElement, [137](#)
  - ForceBasedElement, [184](#)
  - GIFBElement, [203](#)
  - PanelZone, [285](#)
  - SpringBasedElement, [358](#)
- jNode\_ID\_spring
  - SpringBasedElement, [359](#)
- jSpring\_ID
  - SpringBasedElement, [359](#)
- K
  - ModifiedIMK, [269](#)
- K\_combo
  - ConfMander1988Circ, [99](#)
  - ConfMander1988Rect, [122](#)
- K\_factor
  - ModifiedIMK, [269](#)
- Kb
  - Skiadopoulos2021, [340](#)
- Kbf
  - Skiadopoulos2021, [340](#)
- Ke
  - Gupta1999, [238](#)
  - ModifiedIMK, [269](#)
  - Skiadopoulos2021, [340](#)
- ke
  - ConfMander1988Circ, [99](#)
  - ConfMander1988Rect, [123](#)
- Kf
  - Skiadopoulos2021, [340](#)
- Kf\_Ke
  - Skiadopoulos2021, [341](#)
- Kf\_Ke\_tests
  - Skiadopoulos2021, [341](#)
- kg\_unit
  - Units, [61](#)
- kip\_unit
  - Units, [61](#)
- km\_unit
  - Units, [61](#)
- kN\_unit
  - Units, [61](#)
- kNm\_unit
  - Units, [61](#)
- kNmm\_unit
  - Units, [61](#)
- Kp
  - Gupta1999, [238](#)
- kPa\_unit
  - Units, [62](#)
- Ks
  - Skiadopoulos2021, [341](#)
- Ksf
  - Skiadopoulos2021, [341](#)
- ksi\_unit
  - Units, [62](#)
- L
  - GIFBElement, [203](#)
  - ModifiedIMK, [270](#)
  - RCCircShape, [307](#)
  - RCCRectShape, [321](#)

- SteelShape, [376](#)
- L\_0
  - ModifiedIMK, [270](#)
- L\_b
  - ModifiedIMK, [270](#)
- lambda\_i
  - GIFBElement, [204](#)
- lambda\_j
  - GIFBElement, [204](#)
- LateralForce
  - Analysis, [74](#)
- length\_unit
  - Units, [62](#)
- load\_case
  - Analysis, [82](#)
- LoadingProtocol
  - Analysis, [77](#)
- Lp
  - GIFBElement, [204](#)
- M1
  - Skiadopoulos2021, [341](#)
- M1y
  - Gupta1999, [238](#)
- m2\_unit
  - Units, [62](#)
- M2y
  - Gupta1999, [239](#)
- m3\_unit
  - Units, [62](#)
- M3y
  - Gupta1999, [239](#)
- M4
  - Skiadopoulos2021, [341](#)
- m4\_unit
  - Units, [62](#)
- M6
  - Skiadopoulos2021, [342](#)
- m\_unit
  - Units, [63](#)
- master\_node\_ID
  - PanelZone, [285](#)
- mat\_ID
  - PanelZone, [285](#)
- mat\_ID\_i
  - SpringBasedElement, [359](#)
- mat\_ID\_j
  - SpringBasedElement, [359](#)
- MaterialModels, [43](#), [248](#)
  - CheckApplicability, [249](#)
  - Concrete01Funct, [45](#)
  - Concrete04Funct, [47](#)
  - PlotConcrete01, [48](#)
  - PlotConcrete04, [49](#)
- MAX\_ITER
  - Constants, [17](#)
- max\_iter
  - Analysis, [83](#)
  - ForceBasedElement, [184](#)
  - GIFBElement, [204](#)
- MAX\_ITER\_INTEGRATION
  - Constants, [17](#)
- max\_tol
  - GIFBElement, [204](#)
- Mc
  - ModifiedIMK, [270](#)
- McMy
  - ModifiedIMK, [270](#)
- MemberFailure, [249](#)
- MemberModel, [50](#), [250](#)
  - DefinePanelZoneElements, [51](#)
  - DefinePanelZoneNodes, [53](#)
  - Record, [250](#)
  - RecordNodeDef, [251](#)
- mid\_panel\_zone\_height
  - PanelZone, [285](#)
- mid\_panel\_zone\_width
  - PanelZone, [286](#)
- mile\_unit
  - Units, [63](#)
- min\_tol
  - GIFBElement, [204](#)
- min\_unit
  - Units, [63](#)
- mm2\_unit
  - Units, [63](#)
- mm3\_unit
  - Units, [63](#)
- mm4\_unit
  - Units, [63](#)
- mm\_unit
  - Units, [64](#)
- MN\_unit
  - Units, [64](#)
- MNm\_unit
  - Units, [64](#)
- MNmm\_unit
  - Units, [64](#)
- ModifiedIMK, [252](#)
  - \_\_init\_\_, [254](#)
  - a, [267](#)
  - a\_s, [267](#)
  - bf, [267](#)
  - Bilin, [258](#)
  - CheckApplicability, [258](#)
  - Computea, [259](#)
  - Computea\_s, [259](#)
  - ComputeK, [259](#)
  - ComputeKe, [260](#)
  - ComputeMc, [260](#)
  - ComputeMyStar, [261](#)
  - ComputeRefEnergyDissipationCap, [261](#)
  - ComputeTheta\_p, [262](#)
  - ComputeTheta\_pc, [263](#)
  - ComputeTheta\_u, [263](#)
  - ComputeTheta\_y, [264](#)
  - d, [267](#)

- data, 268
- E, 268
- Fy, 268
- gamma\_rm, 268
- h\_1, 268
- ID, 268
- Initialized, 269
- ly\_mod, 269
- iz, 269
- K, 269
- K\_factor, 269
- Ke, 269
- L, 270
- L\_0, 270
- L\_b, 270
- Mc, 270
- McMy, 270
- My, 270
- My\_star, 271
- n, 271
- N\_G, 271
- Npl, 271
- prob\_factor, 271
- rate\_det, 271
- Relnit, 264
- section\_name\_tag, 272
- ShowInfo, 265
- tf, 272
- theta\_p, 272
- theta\_pc, 272
- theta\_u, 272
- theta\_y, 272
- tw, 273
- Type, 273
- UpdateStoredData, 266
- ModifiedIMKSteelShape, 273
  - \_\_init\_\_, 274
  - section, 275
  - section\_name\_tag, 275
- MPa\_unit
  - Units, 64
- My
  - ModifiedIMK, 270
  - SteelShape, 376
- My\_star
  - ModifiedIMK, 271
- N
  - UVC, 410
- n
  - ModifiedIMK, 271
  - SteelShape, 376
- n\_bars
  - FibersCirc, 149
  - RCCircShape, 307
- N\_G
  - ModifiedIMK, 271
- N\_unit
  - Units, 64
- name\_ODB
  - Analysis, 83
- name\_tag
  - RCCircShape, 307
  - RCCRectShape, 321
  - SteelShape, 376
- NegativeValue, 276
- new\_integration\_ID
  - ForceBasedElement, 185
  - GIFBElement, 205
- Nm\_unit
  - Units, 65
- Nmm\_unit
  - Units, 65
- NoApplicability, 276
- node
  - WrongNodeIDConvention, 423
- NodesOrientation
  - FunctionalFeatures, 31
- Npl
  - ModifiedIMK, 271
  - SteelShape, 376
- nr\_bars
  - ConfMander1988Circ, 100
  - ConfMander1988Rect, 123
  - RCCRectShape, 321
- OffsetNodeIDConvention
  - FunctionalFeatures, 32
- Pa\_unit
  - Units, 65
- PanelZone, 277
  - \_\_init\_\_, 278
  - A\_rigid, 283
  - beam\_section\_name\_tag, 283
  - col\_section\_name\_tag, 284
  - CreateMember, 280
  - data, 284
  - E, 284
  - element\_array, 284
  - geo\_transf\_ID, 284
  - I\_rigid, 284
  - Initialized, 285
  - iNode\_ID, 285
  - jNode\_ID, 285
  - master\_node\_ID, 285
  - mat\_ID, 285
  - mid\_panel\_zone\_height, 285
  - mid\_panel\_zone\_width, 286
  - pin\_corners, 286
  - Record, 280
  - RecordNodeDef, 281
  - Relnit, 281
  - ShowInfo, 282
  - spring\_ID, 286
  - UpdateStoredData, 283
- PanelZoneRCS, 286
  - \_\_init\_\_, 287

- beam, 288
- beam\_section\_name\_tag, 288
- col, 288
- col\_section\_name\_tag, 288
- PanelZoneSteelShape, 289
  - \_\_init\_\_, 290
  - beam, 291
  - beam\_section\_name\_tag, 291
  - col, 291
  - col\_section\_name\_tag, 291
- PanelZoneSteelShapeGupta1999, 292
  - \_\_init\_\_, 292
  - beam, 293
  - col, 293
- PanelZoneSteelShapeSkiadopoulos2021, 294
  - \_\_init\_\_, 295
  - beam, 296
  - col, 296
- Pin
  - Connections, 14
- pin\_corners
  - PanelZone, 286
- pinchx
  - Gupta1999, 239
  - Skiadopoulos2021, 342
- pinchy
  - Gupta1999, 239
  - Skiadopoulos2021, 342
- plot\_fiber\_section
  - Fibers, 21
- plot\_member
  - FunctionalFeatures, 33
- plot\_nodes
  - FunctionalFeatures, 34
- PlotConcrete01
  - MaterialModels, 48
- PlotConcrete04
  - MaterialModels, 49
- PositiveValue, 296
- pound\_unit
  - Units, 65
- prob\_factor
  - ModifiedIMK, 271
- ProgressingPercentage
  - FunctionalFeatures, 35
- psi\_unit
  - Units, 65
- Pushover
  - Analysis, 79
- QInf
  - UVC, 410
- r
  - SteelShape, 377
- R0
  - GMP1970, 225
- rBars
  - FibersCirc, 149
- r\_core
  - FibersCirc, 149
- ranges\_y
  - FibersRect, 173
- rate\_det
  - ModifiedIMK, 271
- RCCircShape, 297
  - \_\_init\_\_, 298
  - A, 303
  - Ac, 304
  - As, 304
  - Ay, 304
  - b, 304
  - bc, 304
  - clBars, 304
  - clHoops, 305
  - ComputeEc, 300
  - ComputeI, 301
  - ComputeRhoVol, 301
  - DBars, 305
  - DHoops, 305
  - data, 305
  - e, 305
  - Ec, 305
  - Es, 306
  - Ey, 306
  - fc, 306
  - fs, 306
  - fy, 306
  - I, 306
  - L, 307
  - nBars, 307
  - nameTag, 307
  - ReIinit, 301
  - rhoBars, 307
  - rhoSvol, 307
  - s, 307
  - ShowInfo, 302
  - UpdateStoredData, 303
- RCRectShape, 308
  - \_\_init\_\_, 309
  - A, 317
  - Ac, 317
  - As, 317
  - Ay, 318
  - b, 318
  - bars\_position\_x, 318
  - bars\_ranges\_position\_y, 318
  - bc, 318
  - clBars, 318
  - clHoops, 319
  - ComputeA, 313
  - ComputeAc, 313
  - ComputeEc, 313
  - Computely, 314
  - Computelz, 314
  - ComputeNrBars, 314
  - d, 319



- D\_bars, [319](#)
- D\_hoops, [319](#)
- data, [319](#)
- dc, [319](#)
- e, [320](#)
- Ec, [320](#)
- Es, [320](#)
- Ey, [320](#)
- fc, [320](#)
- fs, [320](#)
- fy, [321](#)
- ly, [321](#)
- lz, [321](#)
- L, [321](#)
- name\_tag, [321](#)
- nr\_bars, [321](#)
- Relnit, [315](#)
- rho\_bars, [322](#)
- rho\_s\_x, [322](#)
- rho\_s\_y, [322](#)
- s, [322](#)
- ShowInfo, [316](#)
- UpdateStoredData, [316](#)
- RCSquareShape, [323](#)
- \_\_init\_\_, [323](#)
- rebarYZ
  - FibersRect, [173](#)
- Record
  - ElasticElement, [133](#)
  - ForceBasedElement, [180](#)
  - GIFBElement, [198](#)
  - MemberModel, [250](#)
  - PanelZone, [280](#)
  - SpringBasedElement, [353](#)
- RecordNodeDef
  - ElasticElement, [134](#)
  - ForceBasedElement, [180](#)
  - GIFBElement, [199](#)
  - MemberModel, [251](#)
  - PanelZone, [281](#)
  - SpringBasedElement, [354](#)
- Relnit
  - ConfMander1988Circ, [93](#)
  - ConfMander1988Rect, [115](#)
  - DataManagement, [128](#)
  - ElasticElement, [134](#)
  - FibersCirc, [144](#)
  - FibersIShape, [156](#)
  - FibersRect, [168](#)
  - ForceBasedElement, [181](#)
  - GIFBElement, [199](#)
  - GMP1970, [221](#)
  - Gupta1999, [233](#)
  - ModifiedIMK, [264](#)
  - PanelZone, [281](#)
  - RCCircShape, [301](#)
  - RCRectShape, [315](#)
  - Skiadopoulos2021, [332](#)
  - SpringBasedElement, [354](#)
  - SteelIShape, [372](#)
  - UnconfMander1988, [384](#)
  - UniaxialBilinear, [397](#)
  - UVC, [406](#)
- rho\_bars
  - RCCircShape, [307](#)
  - RCRectShape, [322](#)
- rho\_cc
  - ConfMander1988Circ, [100](#)
  - ConfMander1988Rect, [123](#)
- rho\_s\_vol
  - ConfMander1988Circ, [100](#)
  - RCCircShape, [307](#)
- rho\_s\_x
  - ConfMander1988Rect, [123](#)
  - RCRectShape, [322](#)
- rho\_s\_y
  - ConfMander1988Rect, [123](#)
  - RCRectShape, [322](#)
- RIGID
  - Constants, [17](#)
- RigidSupport
  - Connections, [14](#)
- RotationalSpring
  - Connections, [15](#)
- Ry
  - Gupta1999, [239](#)
  - Skiadopoulos2021, [342](#)
- s
  - ConfMander1988Circ, [100](#)
  - ConfMander1988Rect, [123](#)
  - RCCircShape, [307](#)
  - RCRectShape, [322](#)
- s\_unit
  - Units, [65](#)
- SaveData
  - DataManagement, [128](#)
- Section, [54](#), [325](#)
  - ComputeACircle, [55](#)
  - ComputeRho, [55](#)
- section
  - ConfMander1988CircRCCircShape, [103](#)
  - ConfMander1988RectRCRectShape, [127](#)
  - ElasticElementSteelIShape, [139](#)
  - FibersCircRCCircShape, [152](#)
  - FibersIShapeSteelIShape, [163](#)
  - FibersRectRCRectShape, [176](#)
  - ForceBasedElementFibersCircRCCircShape, [187](#)
  - ForceBasedElementFibersIShapeSteelIShape, [190](#)
  - ForceBasedElementFibersRectRCRectShape, [193](#)
  - GIFBElementFibersCircRCCircShape, [207](#)
  - GIFBElementFibersRectRCRectShape, [210](#)
  - GIFBElementRCCircShape, [213](#)
  - GIFBElementRCRectShape, [216](#)
  - GMP1970RCRectShape, [227](#)
  - ModifiedIMKSteelIShape, [275](#)

- SpringBasedElementModifiedIMKSteelShape, 362
- SpringBasedElementSteelShape, 365
- UnconfMander1988RCCircShape, 390
- UnconfMander1988RCRectShape, 393
- UniaxialBilinearSteelShape, 402
- UVCCalibratedRCCircShape, 415
- UVCCalibratedRCRectShape, 417
- UVCCalibratedSteelShapeFlange, 419
- UVCCalibratedSteelShapeWeb, 421
- section\_name\_tag
  - ConfMander1988Circ, 100
  - ConfMander1988CircRCCircShape, 103
  - ConfMander1988Rect, 124
  - ConfMander1988RectRCRectShape, 127
  - ElasticElement, 138
  - ElasticElementSteelShape, 140
  - FibersCirc, 149
  - FibersCircRCCircShape, 152
  - FibersIShape, 160
  - FibersIShapeSteelShape, 163
  - FibersRect, 174
  - FibersRectRCRectShape, 176
  - ForceBasedElement, 185
  - ForceBasedElementFibersCircRCCircShape, 187
  - ForceBasedElementFibersIShapeSteelShape, 190
  - ForceBasedElementFibersRectRCRectShape, 193
  - GIFBEelement, 205
  - GIFBEelementFibersCircRCCircShape, 208
  - GIFBEelementFibersRectRCRectShape, 210
  - GIFBEelementRCCircShape, 213
  - GIFBEelementRCRectShape, 216
  - GMP1970, 225
  - GMP1970RCRectShape, 227
  - ModifiedIMK, 272
  - ModifiedIMKSteelShape, 275
  - SpringBasedElement, 359
  - SpringBasedElementModifiedIMKSteelShape, 362
  - SpringBasedElementSteelShape, 365
  - UnconfMander1988, 388
  - UnconfMander1988RCCircShape, 391
  - UnconfMander1988RCRectShape, 393
  - UniaxialBilinear, 400
  - UniaxialBilinearSteelShape, 402
  - UVC, 410
  - UVCCalibratedRCCircShape, 415
  - UVCCalibratedRCRectShape, 417
  - UVCCalibratedSteelShapeFlange, 419
  - UVCCalibratedSteelShapeWeb, 421
- ShowInfo
  - ConfMander1988Circ, 94
  - ConfMander1988Rect, 116
  - DataManagement, 129
  - ElasticElement, 135
  - FibersCirc, 145
  - FibersIShape, 157
  - FibersRect, 169
  - ForceBasedElement, 182
  - GIFBEelement, 200
  - GMP1970, 221
  - Gupta1999, 233
  - ModifiedIMK, 265
  - PanelZone, 282
  - RCCircShape, 302
  - RCRectShape, 316
  - Skiadopoulos2021, 333
  - SpringBasedElement, 355
  - SteelShape, 372
  - UnconfMander1988, 385
  - UniaxialBilinear, 397
  - UVC, 406
- Skiadopoulos2021, 326
  - \_\_init\_\_, 328
  - a\_s, 335
  - beam\_section\_name\_tag, 335
  - beta, 335
  - bf\_c, 335
  - Cf1, 335
  - Cf1\_tests, 336
  - Cf4, 336
  - Cf4\_tests, 336
  - Cf6, 336
  - Cf6\_tests, 336
  - CheckApplicability, 331
  - col\_section\_name\_tag, 336
  - Cw1, 337
  - Cw1\_tests, 337
  - Cw4, 337
  - Cw4\_tests, 337
  - Cw6, 337
  - Cw6\_tests, 337
  - d\_b, 338
  - d\_c, 338
  - data, 338
  - dmg1, 338
  - dmg2, 338
  - E, 338
  - Fy, 339
  - G, 339
  - Gamma\_1, 339
  - Gamma\_4, 339
  - Gamma\_6, 339
  - Hysteretic, 332
  - I\_c, 339
  - ID, 340
  - Initialized, 340
  - Kb, 340
  - Kbf, 340
  - Ke, 340
  - Kf, 340
  - Kf\_Ke, 341
  - Kf\_Ke\_tests, 341
  - Ks, 341
  - Ksf, 341

- M1, [341](#)
- M4, [341](#)
- M6, [342](#)
- pinchx, [342](#)
- pinchy, [342](#)
- Relnit, [332](#)
- Ry, [342](#)
- ShowInfo, [333](#)
- t\_dp, [342](#)
- t\_fbp, [342](#)
- t\_p, [343](#)
- t\_pz, [343](#)
- tf\_b, [343](#)
- tf\_c, [343](#)
- UpdateStoredData, [334](#)
- V1, [343](#)
- V4, [343](#)
- V6, [344](#)
- Skiadopoulos2021RCS, [344](#)
  - \_\_init\_\_, [345](#)
  - beam, [346](#)
  - beam\_section\_name\_tag, [346](#)
- Skiadopoulos2021SteelShape, [347](#)
  - \_\_init\_\_, [348](#)
  - beam, [349](#)
  - beam\_section\_name\_tag, [349](#)
  - col, [349](#)
  - col\_section\_name\_tag, [349](#)
- spring\_ID
  - PanelZone, [286](#)
- SpringBasedElement, [350](#)
  - \_\_init\_\_, [351](#)
  - A, [356](#)
  - CreateMember, [353](#)
  - data, [357](#)
  - E, [357](#)
  - ele\_orientation, [357](#)
  - element\_array, [357](#)
  - element\_ID, [357](#)
  - geo\_transf\_ID, [357](#)
  - Initialized, [358](#)
  - iNode\_ID, [358](#)
  - iNode\_ID\_spring, [358](#)
  - iSpring\_ID, [358](#)
  - ly\_mod, [358](#)
  - jNode\_ID, [358](#)
  - jNode\_ID\_spring, [359](#)
  - jSpring\_ID, [359](#)
  - mat\_ID\_i, [359](#)
  - mat\_ID\_j, [359](#)
  - Record, [353](#)
  - RecordNodeDef, [354](#)
  - Relnit, [354](#)
  - section\_name\_tag, [359](#)
  - ShowInfo, [355](#)
  - UpdateStoredData, [356](#)
- SpringBasedElementModifiedIMKSteelShape, [360](#)
  - \_\_init\_\_, [360](#)
  - section, [362](#)
  - section\_name\_tag, [362](#)
- SpringBasedElementSteelShape, [363](#)
  - \_\_init\_\_, [364](#)
  - section, [365](#)
  - section\_name\_tag, [365](#)
- Steel01
  - UniaxialBilinear, [398](#)
- Steel02
  - GMP1970, [222](#)
- SteelShape, [365](#)
  - \_\_init\_\_, [367](#)
  - A, [374](#)
  - bf, [374](#)
  - Compute\_iy, [369](#)
  - Compute\_iz, [369](#)
  - ComputeA, [369](#)
  - Computely, [370](#)
  - Computelz, [370](#)
  - ComputeWply, [371](#)
  - ComputeWplz, [371](#)
  - d, [374](#)
  - data, [374](#)
  - E, [374](#)
  - Fy, [374](#)
  - Fy\_web, [375](#)
  - h\_1, [375](#)
  - ly, [375](#)
  - iy, [375](#)
  - ly\_mod, [375](#)
  - lz, [375](#)
  - iz, [376](#)
  - L, [376](#)
  - My, [376](#)
  - n, [376](#)
  - name\_tag, [376](#)
  - Npl, [376](#)
  - r, [377](#)
  - Relnit, [372](#)
  - ShowInfo, [372](#)
  - tf, [377](#)
  - tw, [377](#)
  - Type, [377](#)
  - UpdateStoredData, [373](#)
  - Wply, [377](#)
  - Wplz, [377](#)
- t\_dp
  - Gupta1999, [239](#)
  - Skiadopoulos2021, [342](#)
- t\_fbp
  - Skiadopoulos2021, [342](#)
- t\_p
  - Gupta1999, [240](#)
  - Skiadopoulos2021, [343](#)
- t\_pz
  - Gupta1999, [240](#)
  - Skiadopoulos2021, [343](#)
- t\_unit

- Units, 66
- test\_opt
  - Analysis, 83
- test\_type
  - Analysis, 83
- tf
  - ModifiedIMK, 272
  - SteelShape, 377
- tf\_b
  - FibersIShape, 160
  - Gupta1999, 240
  - Skiadopoulos2021, 343
- tf\_c
  - Gupta1999, 240
  - Skiadopoulos2021, 343
- tf\_t
  - FibersIShape, 160
- theta\_p
  - ModifiedIMK, 272
- theta\_pc
  - ModifiedIMK, 272
- theta\_u
  - ModifiedIMK, 272
- theta\_y
  - ModifiedIMK, 272
- time\_unit
  - Units, 66
- TOL
  - Constants, 17
- tol
  - Analysis, 83
  - ForceBasedElement, 185
- TOL\_INTEGRATION
  - Constants, 17
- top\_flange\_mat\_ID
  - FibersIShape, 161
- tw
  - FibersIShape, 161
  - ModifiedIMK, 273
  - SteelShape, 377
- Type
  - ModifiedIMK, 273
  - SteelShape, 377
- unconf\_mat\_ID
  - FibersCirc, 150
  - FibersRect, 174
- UnconfMander1988, 378
  - \_\_init\_\_, 379
  - beta, 386
  - CheckApplicability, 381
  - Compute\_ec, 381
  - Compute\_ecp, 381
  - Compute\_ecu, 382
  - Compute\_et, 382
  - Compute\_fct, 383
  - Concrete01, 383
  - Concrete04, 384
  - data, 387
  - Ec, 387
  - ec, 387
  - ecp, 387
  - ecu, 387
  - et, 387
  - fc, 388
  - fct, 388
  - ID, 388
  - Initialized, 388
  - Relnit, 384
  - section\_name\_tag, 388
  - ShowInfo, 385
  - UpdateStoredData, 386
- UnconfMander1988RCCircShape, 389
  - \_\_init\_\_, 389
  - section, 390
  - section\_name\_tag, 391
- UnconfMander1988RCRectShape, 391
  - \_\_init\_\_, 392
  - section, 393
  - section\_name\_tag, 393
- UniaxialBilinear, 393
  - \_\_init\_\_, 394
  - b, 399
  - CheckApplicability, 396
  - data, 399
  - Ey, 399
  - ey, 399
  - fy, 400
  - ID, 400
  - Initialized, 400
  - Relnit, 397
  - section\_name\_tag, 400
  - ShowInfo, 397
  - Steel01, 398
  - UpdateStoredData, 398
- UniaxialBilinearSteelShape, 401
  - \_\_init\_\_, 401
  - section, 402
  - section\_name\_tag, 402
- Units, 56
  - cm2\_unit, 57
  - cm3\_unit, 57
  - cm4\_unit, 58
  - cm\_unit, 58
  - dm2\_unit, 58
  - dm3\_unit, 58
  - dm4\_unit, 58
  - dm\_unit, 58
  - force\_unit, 59
  - ft2\_unit, 59
  - ft3\_unit, 59
  - ft4\_unit, 59
  - ft\_unit, 59
  - GN\_unit, 59
  - GPa\_unit, 60
  - hours\_unit, 60
  - inch2\_unit, 60

- inch3\_unit, [60](#)
- inch4\_unit, [60](#)
- inch\_unit, [60](#)
- kg\_unit, [61](#)
- kip\_unit, [61](#)
- km\_unit, [61](#)
- kN\_unit, [61](#)
- kNm\_unit, [61](#)
- kNmm\_unit, [61](#)
- kPa\_unit, [62](#)
- ksi\_unit, [62](#)
- length\_unit, [62](#)
- m2\_unit, [62](#)
- m3\_unit, [62](#)
- m4\_unit, [62](#)
- m\_unit, [63](#)
- mile\_unit, [63](#)
- min\_unit, [63](#)
- mm2\_unit, [63](#)
- mm3\_unit, [63](#)
- mm4\_unit, [63](#)
- mm\_unit, [64](#)
- MN\_unit, [64](#)
- MNm\_unit, [64](#)
- MNmm\_unit, [64](#)
- MPa\_unit, [64](#)
- N\_unit, [64](#)
- Nm\_unit, [65](#)
- Nmm\_unit, [65](#)
- Pa\_unit, [65](#)
- pound\_unit, [65](#)
- psi\_unit, [65](#)
- s\_unit, [65](#)
- t\_unit, [66](#)
- time\_unit, [66](#)
- UpdateStoredData
  - ConfMander1988Circ, [95](#)
  - ConfMander1988Rect, [117](#)
  - DataManagement, [130](#)
  - ElasticElement, [135](#)
  - FibersCirc, [146](#)
  - FibersIShape, [158](#)
  - FibersRect, [170](#)
  - ForceBasedElement, [182](#)
  - GIFBElement, [201](#)
  - GMP1970, [222](#)
  - Gupta1999, [234](#)
  - ModifiedIMK, [266](#)
  - PanelZone, [283](#)
  - RCCircShape, [303](#)
  - RCCRectShape, [316](#)
  - Skiadopoulos2021, [334](#)
  - SpringBasedElement, [356](#)
  - SteelShape, [373](#)
  - UnconfMander1988, [386](#)
  - UniaxialBilinear, [398](#)
  - UVC, [407](#)
- UVC, [403](#)
- \_\_init\_\_, [404](#)
- a, [408](#)
- b, [408](#)
- CheckApplicability, [405](#)
- cK, [408](#)
- data, [408](#)
- DInf, [409](#)
- Ey, [409](#)
- fy, [409](#)
- gammaK, [409](#)
- ID, [409](#)
- Initialized, [409](#)
- N, [410](#)
- QInf, [410](#)
- Relnit, [406](#)
- section\_name\_tag, [410](#)
- ShowInfo, [406](#)
- UpdateStoredData, [407](#)
- UVCuniaxial, [407](#)
- UVCCalibrated, [410](#)
- \_\_init\_\_, [411](#)
- calibration, [413](#)
- UVCCalibratedRCCircShape, [414](#)
- \_\_init\_\_, [414](#)
- section, [415](#)
- section\_name\_tag, [415](#)
- UVCCalibratedRCRectShape, [416](#)
- \_\_init\_\_, [416](#)
- section, [417](#)
- section\_name\_tag, [417](#)
- UVCCalibratedSteelShapeFlange, [418](#)
- \_\_init\_\_, [418](#)
- section, [419](#)
- section\_name\_tag, [419](#)
- UVCCalibratedSteelShapeWeb, [420](#)
- \_\_init\_\_, [420](#)
- section, [421](#)
- section\_name\_tag, [421](#)
- UVCuniaxial
  - UVC, [407](#)
- V1
  - Skiadopoulos2021, [343](#)
- V4
  - Skiadopoulos2021, [343](#)
- V6
  - Skiadopoulos2021, [344](#)
- Vy
  - Gupta1999, [240](#)
- web\_mat\_ID
  - FibersIShape, [161](#)
- Wply
  - SteelShape, [377](#)
- Wplz
  - SteelShape, [377](#)
- WrongArgument, [422](#)
- WrongDimension, [422](#)
- WrongNodeIDConvention, [423](#)

- [\\_\\_init\\_\\_, 423](#)
  - [node, 423](#)
- [wx\\_bottom](#)
  - [ConfMander1988Rect, 124](#)
- [wx\\_top](#)
  - [ConfMander1988Rect, 124](#)
- [wy](#)
  - [ConfMander1988Rect, 124](#)
- [ZERO](#)
  - [Constants, 18](#)
- [ZeroDivision, 424](#)
- [ZeroLength, 424](#)
  - [\\_\\_init\\_\\_, 425](#)
  - [element, 425](#)