DGOSPREY

Version 0.0 beta

Generated by Doxygen 1.8.3.1

Mon Feb 1 2016 14:34:47

Contents

1 Introduction 1

1.1 Copyright Statement . . . 1

1.2 General Information . . . 1

2 Hierarchical Index 1

2.1 Class Hierarchy . . . 1

3 Class Index 3

3.1 Class List . . . 4

4 File Index 7

4.1 File List . . . 7

5 Class Documentation 10

5.1 AdsorbentProperties Class Reference . . . 10

5.1.1 Detailed Description . . . 11

5.1.2 Constructor & Destructor Documentation . . . 11

5.1.3 Member Function Documentation . . . 11

5.1.4 Member Data Documentation . . . 11

5.2 AdsorptionHeatAccumulation Class Reference . . . 12

5.2.1 Detailed Description . . . 13

5.2.2 Constructor & Destructor Documentation . . . 13

5.2.3 Member Function Documentation . . . 13

5.2.4 Member Data Documentation . . . 14

5.3 AdsorptionMassTransfer Class Reference . . . 14

5.3.1 Detailed Description . . . 15

5.3.2 Constructor & Destructor Documentation . . . 15

5.3.3 Member Function Documentation . . . 15

5.3.4 Member Data Documentation . . . 15

5.4 ARNOLDI_DATA Struct Reference . . . 16

5.4.1 Detailed Description . . . 16

5.4.2 Member Data Documentation . . . 17

5.5 BACKTRACK_DATA Struct Reference . . . 18

5.5.1 Detailed Description . . . 18

5.5.2 Member Data Documentation . . . 18

5.6 BedHeatAccumulation Class Reference . . . 19

5.6.1 Detailed Description . . . 20

5.6.2 Constructor & Destructor Documentation . . . 20

5.6.3 Member Function Documentation . . . 20

5.6.4 Member Data Documentation . . . 20

5.7 BedMassAccumulation Class Reference . . . 21
5.7.1 Detailed Description . . . 21
5.7.2 Constructor & Destructor Documentation . . . 21
5.7.3 Member Function Documentation . . . 22
5.7.4 Member Data Documentation . . . 22
5.8 BedProperties Class Reference . . . 22
5.8.1 Detailed Description . . . 23
5.8.2 Constructor & Destructor Documentation . . . 24
5.8.3 Member Function Documentation . . . 24
5.8.4 Member Data Documentation . . . 24
5.9 BedWallHeatTransfer Class Reference . . . 26
5.9.1 Detailed Description . . . 26
5.9.2 Constructor & Destructor Documentation . . . 27
5.9.3 Member Function Documentation . . . 27
5.9.4 Member Data Documentation . . . 27
5.10 BiCGSTAB_DATA Struct Reference . . . 27
5.10.1 Detailed Description . . . 29
5.10.2 Member Data Documentation . . . 29
5.11 CGS_DATA Struct Reference . . . 31
5.11.1 Detailed Description . . . 32
5.11.2 Member Data Documentation . . . 32
5.12 ColumnTemperatureIC Class Reference . . . 35
5.12.1 Detailed Description . . . 35
5.12.2 Constructor & Destructor Documentation . . . 35
5.12.3 Member Function Documentation . . . 36
5.12.4 Member Data Documentation . . . 36
5.13 ConcentrationIC Class Reference . . . 36
5.13.1 Detailed Description . . . 36
5.13.2 Constructor & Destructor Documentation . . . 37
5.13.3 Member Function Documentation . . . 37
5.13.4 Member Data Documentation . . . 37
5.14 CoupledLDF Class Reference . . . 37
5.14.1 Detailed Description . . . 38
5.14.2 Constructor & Destructor Documentation . . . 39
5.14.3 Member Function Documentation . . . 39
5.14.4 Member Data Documentation . . . 39
5.15 DGAdvection Class Reference . . . 40
5.15.1 Detailed Description . . . 40
5.15.2 Constructor & Destructor Documentation . . . 41
5.15.3 Member Function Documentation . . . 41

5.15.4 Member Data Documentation . . . 41
5.16 DGAnisotropicDiffusion Class Reference . . . 42
5.16.1 Detailed Description . . . 42
5.16.2 Constructor & Destructor Documentation . . . 43
5.16.3 Member Function Documentation . . . 43
5.16.4 Member Data Documentation . . . 43
5.17 DGColumnHeatAdvection Class Reference . . . 44
5.17.1 Detailed Description . . . 45
5.17.2 Constructor & Destructor Documentation . . . 45
5.17.3 Member Function Documentation . . . 45
5.17.4 Member Data Documentation . . . 46
5.18 DGColumnHeatDispersion Class Reference . . . 46
5.18.1 Detailed Description . . . 47
5.18.2 Constructor & Destructor Documentation . . . 48
5.18.3 Member Function Documentation . . . 48
5.18.4 Member Data Documentation . . . 48
5.19 DGColumnMassAdvection Class Reference . . . 49
5.19.1 Detailed Description . . . 50
5.19.2 Constructor & Destructor Documentation . . . 50
5.19.3 Member Function Documentation . . . 50
5.19.4 Member Data Documentation . . . 51
5.20 DGColumnMassDispersion Class Reference . . . 51
5.20.1 Detailed Description . . . 52
5.20.2 Constructor & Destructor Documentation . . . 53
5.20.3 Member Function Documentation . . . 53
5.20.4 Member Data Documentation . . . 53
5.21 DGColumnWallHeatFluxBC Class Reference . . . 54
5.21.1 Detailed Description . . . 55
5.21.2 Constructor & Destructor Documentation . . . 56
5.21.3 Member Function Documentation . . . 56
5.21.4 Member Data Documentation . . . 56
5.22 DGColumnWallHeatFluxLimitedBC Class Reference . . . 57
5.22.1 Detailed Description . . . 59
5.22.2 Constructor & Destructor Documentation . . . 59
5.22.3 Member Function Documentation . . . 59
5.22.4 Member Data Documentation . . . 59
5.23 DGFluxBC Class Reference . . . 61
5.23.1 Detailed Description . . . 62
5.23.2 Constructor & Destructor Documentation . . . 62
5.23.3 Member Function Documentation . . . 62

5.23.4 Member Data Documentation . . . 62
5.24 DGFluxLimitedBC Class Reference . . . 63
5.24.1 Detailed Description . . . 64
5.24.2 Constructor & Destructor Documentation . . . 65
5.24.3 Member Function Documentation . . . 65
5.24.4 Member Data Documentation . . . 65
5.25 DGHeatFluxBC Class Reference . . . 66
5.25.1 Detailed Description . . . 68
5.25.2 Constructor & Destructor Documentation . . . 68
5.25.3 Member Function Documentation . . . 68
5.25.4 Member Data Documentation . . . 68
5.26 DGHeatFluxLimitedBC Class Reference . . . 70
5.26.1 Detailed Description . . . 71
5.26.2 Constructor & Destructor Documentation . . . 71
5.26.3 Member Function Documentation . . . 71
5.26.4 Member Data Documentation . . . 72
5.27 DGMassFluxBC Class Reference . . . 73
5.27.1 Detailed Description . . . 75
5.27.2 Constructor & Destructor Documentation . . . 75
5.27.3 Member Function Documentation . . . 75
5.27.4 Member Data Documentation . . . 75
5.28 DGMassFluxLimitedBC Class Reference . . . 77
5.28.1 Detailed Description . . . 79
5.28.2 Constructor & Destructor Documentation . . . 79
5.28.3 Member Function Documentation . . . 79
5.28.4 Member Data Documentation . . . 79
5.29 DgospreyApp Class Reference . . . 81
5.29.1 Detailed Description . . . 82
5.29.2 Constructor & Destructor Documentation . . . 82
5.29.3 Member Function Documentation . . . 82
5.30 FINCH_DATA Struct Reference . . . 82
5.30.1 Detailed Description . . . 87
5.30.2 Member Data Documentation . . . 87
5.31 FlowProperties Class Reference . . . 96
5.31.1 Detailed Description . . . 98
5.31.2 Constructor & Destructor Documentation . . . 98
5.31.3 Member Function Documentation . . . 98
5.31.4 Member Data Documentation . . . 98
5.32 GAdvection Class Reference . . . 101
5.32.1 Detailed Description . . . 101

5.32.2 Constructor & Destructor Documentation . . . 102
5.32.3 Member Function Documentation . . . 102
5.32.4 Member Data Documentation . . . 102
5.33 GAnisotropicDiffusion Class Reference . . . 103
5.33.1 Detailed Description . . . 103
5.33.2 Constructor & Destructor Documentation . . . 104
5.33.3 Member Function Documentation . . . 104
5.33.4 Member Data Documentation . . . 104
5.34 GColumnHeatAdvection Class Reference . . . 105
5.34.1 Detailed Description . . . 106
5.34.2 Constructor & Destructor Documentation . . . 106
5.34.3 Member Function Documentation . . . 106
5.34.4 Member Data Documentation . . . 106
5.35 GColumnHeatDispersion Class Reference . . . 107
5.35.1 Detailed Description . . . 108
5.35.2 Constructor & Destructor Documentation . . . 108
5.35.3 Member Function Documentation . . . 108
5.35.4 Member Data Documentation . . . 109
5.36 GColumnMassAdvection Class Reference . . . 109
5.36.1 Detailed Description . . . 110
5.36.2 Constructor & Destructor Documentation . . . 110
5.36.3 Member Function Documentation . . . 111
5.36.4 Member Data Documentation . . . 111
5.37 GColumnMassDispersion Class Reference . . . 111
5.37.1 Detailed Description . . . 113
5.37.2 Constructor & Destructor Documentation . . . 113
5.37.3 Member Function Documentation . . . 113
5.37.4 Member Data Documentation . . . 113
5.38 GCR_DATA Struct Reference . . . 114
5.38.1 Detailed Description . . . 115
5.38.2 Member Data Documentation . . . 115
5.39 GMRESLP_DATA Struct Reference . . . 118
5.39.1 Detailed Description . . . 118
5.39.2 Member Data Documentation . . . 119
5.40 GMRESR_DATA Struct Reference . . . 120
5.40.1 Detailed Description . . . 121
5.40.2 Member Data Documentation . . . 121
5.41 GMRESRP_DATA Struct Reference . . . 123
5.41.1 Detailed Description . . . 124
5.41.2 Member Data Documentation . . . 124

5.42 GPAST_DATA Struct Reference . . . 127
5.42.1 Detailed Description . . . 127
5.42.2 Member Data Documentation . . . 128
5.43 GSTA_DATA Struct Reference . . . 129
5.43.1 Detailed Description . . . 129
5.43.2 Member Data Documentation . . . 129
5.44 KMS_DATA Struct Reference . . . 129
5.44.1 Detailed Description . . . 130
5.44.2 Member Data Documentation . . . 131
5.45 LinearDrivingForce Class Reference . . . 132
5.45.1 Detailed Description . . . 133
5.45.2 Constructor & Destructor Documentation . . . 133
5.45.3 Member Function Documentation . . . 133
5.45.4 Member Data Documentation . . . 134
5.46 MAGPIE_Adsorption Class Reference . . . 134
5.46.1 Detailed Description . . . 135
5.46.2 Constructor & Destructor Documentation . . . 135
5.46.3 Member Function Documentation . . . 135
5.46.4 Member Data Documentation . . . 135
5.47 MAGPIE_AdsorptionHeat Class Reference . . . 135
5.47.1 Detailed Description . . . 136
5.47.2 Constructor & Destructor Documentation . . . 136
5.47.3 Member Function Documentation . . . 136
5.47.4 Member Data Documentation . . . 137
5.48 MAGPIE_DATA Struct Reference . . . 137
5.48.1 Detailed Description . . . 137
5.48.2 Member Data Documentation . . . 137
5.49 MAGPIE_Perturbation Class Reference . . . 138
5.49.1 Detailed Description . . . 138
5.49.2 Constructor & Destructor Documentation . . . 138
5.49.3 Member Function Documentation . . . 139
5.49.4 Member Data Documentation . . . 139
5.50 MagpieAdsorbateProperties Class Reference . . . 139
5.50.1 Detailed Description . . . 140
5.50.2 Constructor & Destructor Documentation . . . 141
5.50.3 Member Function Documentation . . . 141
5.50.4 Member Data Documentation . . . 141
5.51 Matrix< T > Class Template Reference . . . 143
5.51.1 Detailed Description . . . 146
5.51.2 Constructor & Destructor Documentation . . . 146

5.51.3 Member Function Documentation . . . 146
5.51.4 Member Data Documentation . . . 153
5.52 MIXED_GAS Struct Reference . . . 154
5.52.1 Detailed Description . . . 155
5.52.2 Member Data Documentation . . . 155
5.53 mSPD_DATA Struct Reference . . . 157
5.53.1 Detailed Description . . . 157
5.53.2 Member Data Documentation . . . 157
5.54 NUM_JAC_DATA Struct Reference . . . 158
5.54.1 Detailed Description . . . 158
5.54.2 Member Data Documentation . . . 158
5.55 OPTRANS_DATA Struct Reference . . . 159
5.55.1 Detailed Description . . . 159
5.55.2 Member Data Documentation . . . 159
5.56 PCG_DATA Struct Reference . . . 159
5.56.1 Detailed Description . . . 160
5.56.2 Member Data Documentation . . . 160
5.57 PICARD_DATA Struct Reference . . . 162
5.57.1 Detailed Description . . . 163
5.57.2 Member Data Documentation . . . 163
5.58 PJFNK_DATA Struct Reference . . . 164
5.58.1 Detailed Description . . . 166
5.58.2 Member Data Documentation . . . 166
5.59 PURE_GAS Struct Reference . . . 169
5.59.1 Detailed Description . . . 170
5.59.2 Member Data Documentation . . . 170
5.60 SCOPSOWL_DATA Struct Reference . . . 171
5.60.1 Detailed Description . . . 173
5.60.2 Member Data Documentation . . . 173
5.61 SCOPSOWL_PARAM_DATA Struct Reference . . . 177
5.61.1 Detailed Description . . . 178
5.61.2 Member Data Documentation . . . 178
5.62 SKUA_DATA Struct Reference . . . 180
5.62.1 Detailed Description . . . 181
5.62.2 Member Data Documentation . . . 181
5.63 SKUA_PARAM Struct Reference . . . 184
5.63.1 Detailed Description . . . 184
5.63.2 Member Data Documentation . . . 184
5.64 SYSTEM_DATA Struct Reference . . . 185
5.64.1 Detailed Description . . . 186

5.64.2 Member Data Documentation . . . 186
5.65 TotalColumnPressure Class Reference . . . 188
5.65.1 Detailed Description . . . 189
5.65.2 Constructor & Destructor Documentation . . . 189
5.65.3 Member Function Documentation . . . 189
5.65.4 Member Data Documentation . . . 189
5.66 TotalPressureIC Class Reference . . . 189
5.66.1 Detailed Description . . . 190
5.66.2 Constructor & Destructor Documentation . . . 190
5.66.3 Member Function Documentation . . . 190
5.66.4 Member Data Documentation . . . 190
5.67 WallAmbientHeatTransfer Class Reference . . . 191
5.67.1 Detailed Description . . . 191
5.67.2 Constructor & Destructor Documentation . . . 192
5.67.3 Member Function Documentation . . . 192
5.67.4 Member Data Documentation . . . 192
5.68 WallHeatAccumulation Class Reference . . . 192
5.68.1 Detailed Description . . . 193
5.68.2 Constructor & Destructor Documentation . . . 193
5.68.3 Member Function Documentation . . . 193
5.68.4 Member Data Documentation . . . 194

6 File Documentation **194**

6.1 AdsorbentProperties.h File Reference . . . 194
6.1.1 Detailed Description . . . 194
6.1.2 Function Documentation . . . 195
6.2 AdsorptionHeatAccumulation.h File Reference . . . 195
6.2.1 Detailed Description . . . 196
6.2.2 Function Documentation . . . 196
6.3 AdsorptionMassTransfer.h File Reference . . . 196
6.3.1 Detailed Description . . . 197
6.3.2 Function Documentation . . . 197
6.4 BedHeatAccumulation.h File Reference . . . 197
6.4.1 Detailed Description . . . 197
6.4.2 Function Documentation . . . 198
6.5 BedMassAccumulation.h File Reference . . . 198
6.5.1 Detailed Description . . . 198
6.5.2 Function Documentation . . . 199
6.6 BedProperties.h File Reference . . . 199
6.6.1 Function Documentation . . . 199

6.7 BedWallHeatTransfer.h File Reference . . . 199
6.7.1 Detailed Description . . . 200
6.7.2 Function Documentation . . . 200
6.8 ColumnTemperatureIC.h File Reference . . . 200
6.8.1 Detailed Description . . . 200
6.8.2 Function Documentation . . . 201
6.9 ConcentrationIC.h File Reference . . . 201
6.9.1 Detailed Description . . . 201
6.9.2 Function Documentation . . . 202
6.10 CoupledLDF.h File Reference . . . 202
6.10.1 Detailed Description . . . 202
6.10.2 Macro Definition Documentation . . . 203
6.10.3 Function Documentation . . . 203
6.11 DGAdvection.h File Reference . . . 203
6.11.1 Detailed Description . . . 203
6.11.2 Function Documentation . . . 204
6.12 DGAnisotropicDiffusion.h File Reference . . . 204
6.12.1 Detailed Description . . . 204
6.12.2 Function Documentation . . . 205
6.13 DGColumnHeatAdvection.h File Reference . . . 205
6.13.1 Detailed Description . . . 206
6.13.2 Macro Definition Documentation . . . 206
6.13.3 Function Documentation . . . 206
6.14 DGColumnHeatDispersion.h File Reference . . . 206
6.14.1 Detailed Description . . . 207
6.14.2 Macro Definition Documentation . . . 207
6.14.3 Function Documentation . . . 207
6.15 DGColumnMassAdvection.h File Reference . . . 208
6.15.1 Detailed Description . . . 208
6.15.2 Function Documentation . . . 208
6.16 DGColumnMassDispersion.h File Reference . . . 209
6.16.1 Detailed Description . . . 209
6.16.2 Function Documentation . . . 210
6.17 DGColumnWallHeatFluxBC.h File Reference . . . 210
6.17.1 Detailed Description . . . 210
6.17.2 Function Documentation . . . 210
6.18 DGColumnWallHeatFluxLimitedBC.h File Reference . . . 211
6.18.1 Detailed Description . . . 211
6.18.2 Function Documentation . . . 211
6.19 DGFluxBC.h File Reference . . . 211

6.19.1 Detailed Description . . . 212
6.19.2 Function Documentation . . . 212
6.20 DGFluxLimitedBC.h File Reference . . . 212
6.20.1 Detailed Description . . . 213
6.20.2 Function Documentation . . . 213
6.21 DGHeatFluxBC.h File Reference . . . 213
6.21.1 Detailed Description . . . 214
6.21.2 Function Documentation . . . 214
6.22 DGHeatFluxLimitedBC.h File Reference . . . 214
6.22.1 Detailed Description . . . 215
6.22.2 Function Documentation . . . 215
6.23 DGMassFluxBC.h File Reference . . . 215
6.23.1 Detailed Description . . . 215
6.23.2 Function Documentation . . . 216
6.24 DGMassFluxLimitedBC.h File Reference . . . 216
6.24.1 Detailed Description . . . 216
6.24.2 Function Documentation . . . 217
6.25 DgospreyApp.h File Reference . . . 217
6.25.1 Detailed Description . . . 217
6.25.2 Function Documentation . . . 218
6.26 DgospreyRevision.h File Reference . . . 218
6.26.1 Macro Definition Documentation . . . 218
6.27 DgospreyRevision.h File Reference . . . 218
6.27.1 Macro Definition Documentation . . . 218
6.28 egret.h File Reference . . . 218
6.28.1 Detailed Description . . . 220
6.28.2 Macro Definition Documentation . . . 220
6.28.3 Function Documentation . . . 222
6.29 error.h File Reference . . . 222
6.29.1 Detailed Description . . . 223
6.29.2 Macro Definition Documentation . . . 223
6.29.3 Enumeration Type Documentation . . . 224
6.29.4 Function Documentation . . . 225
6.30 finch.h File Reference . . . 225
6.30.1 Detailed Description . . . 227
6.30.2 Enumeration Type Documentation . . . 228
6.30.3 Function Documentation . . . 228
6.31 flock.h File Reference . . . 232
6.31.1 Detailed Description . . . 232
6.32 FlowProperties.h File Reference . . . 232

6.32.1 Detailed Description . . . 233
6.32.2 Macro Definition Documentation . . . 233
6.32.3 Function Documentation . . . 234
6.33 GAdvection.h File Reference . . . 234
6.33.1 Detailed Description . . . 234
6.33.2 Function Documentation . . . 234
6.34 GAnisotropicDiffusion.h File Reference . . . 234
6.34.1 Detailed Description . . . 235
6.34.2 Function Documentation . . . 235
6.35 GColumnHeatAdvection.h File Reference . . . 235
6.35.1 Detailed Description . . . 236
6.35.2 Function Documentation . . . 236
6.36 GColumnHeatDispersion.h File Reference . . . 236
6.36.1 Detailed Description . . . 236
6.36.2 Function Documentation . . . 237
6.37 GColumnMassAdvection.h File Reference . . . 237
6.37.1 Detailed Description . . . 237
6.37.2 Function Documentation . . . 238
6.38 GColumnMassDispersion.h File Reference . . . 238
6.38.1 Detailed Description . . . 238
6.38.2 Function Documentation . . . 239
6.39 lark.h File Reference . . . 239
6.39.1 Detailed Description . . . 241
6.39.2 Macro Definition Documentation . . . 242
6.39.3 Enumeration Type Documentation . . . 242
6.39.4 Function Documentation . . . 243
6.40 LinearDrivingForce.h File Reference . . . 253
6.40.1 Detailed Description . . . 254
6.40.2 Function Documentation . . . 254
6.41 macaw.h File Reference . . . 254
6.41.1 Detailed Description . . . 255
6.41.2 Macro Definition Documentation . . . 255
6.42 magpie.h File Reference . . . 255
6.42.1 Detailed Description . . . 257
6.42.2 Macro Definition Documentation . . . 258
6.42.3 Function Documentation . . . 259
6.43 MAGPIE_Adsorption.h File Reference . . . 263
6.43.1 Detailed Description . . . 263
6.43.2 Function Documentation . . . 264
6.44 MAGPIE_AdsorptionHeat.h File Reference . . . 264

6.44.1 Detailed Description . . . 264
6.44.2 Function Documentation . . . 265
6.45 MAGPIE_Perturbation.h File Reference . . . 265
6.45.1 Detailed Description . . . 265
6.45.2 Function Documentation . . . 266
6.46 MagpieAdsorbateProperties.h File Reference . . . 266
6.46.1 Detailed Description . . . 266
6.46.2 Function Documentation . . . 267
6.47 scopsowl.h File Reference . . . 267
6.47.1 Detailed Description . . . 268
6.47.2 Macro Definition Documentation . . . 269
6.47.3 Function Documentation . . . 269
6.48 skua.h File Reference . . . 273
6.48.1 Detailed Description . . . 275
6.48.2 Macro Definition Documentation . . . 275
6.48.3 Function Documentation . . . 276
6.49 TotalColumnPressure.h File Reference . . . 279
6.49.1 Detailed Description . . . 279
6.49.2 Function Documentation . . . 280
6.50 TotalPressureIC.h File Reference . . . 280
6.50.1 Detailed Description . . . 280
6.50.2 Function Documentation . . . 281
6.51 WallAmbientHeatTransfer.h File Reference . . . 281
6.51.1 Detailed Description . . . 281
6.51.2 Function Documentation . . . 282
6.52 WallHeatAccumulation.h File Reference . . . 282
6.52.1 Detailed Description . . . 282
6.52.2 Function Documentation . . . 283

Index **283**

1 Introduction

1.1 Copyright Statement

Copyright

1.2 General Information

DGOSPREY is a MOOSE based application designed to simulate mass and energy transport of gases through a packed-bed column reactor. It uses Discontinuous Galerkin (DG) Finite Element Methods (FEM) to ensure conservation of mass and energy are maintained throughout the entire domain, and each individual sub-domain of the problem. There are currently no slope limiter methods available in the MOOSE framework, so to prevent overshoot and undershoot oscillations we use monomial shape functions for the non-linear variables.

Warning

This is an unfinished application. Use with caution.

2 Hierarchical Index

2.1 Class Hierarchy

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

ARNOLDI_DATA 16
AuxKernel
- **MAGPIE_Adsorption** 134
- **MAGPIE_AdsorptionHeat** 135
- **MAGPIE_Perturbation** 138
- **TotalColumnPressure** 188

BACKTRACK_DATA 18

BiCGSTAB_DATA 27

CGS_DATA 31
DGKernel
- **DGAdvection** 40
  - **DGColumnHeatAdvection** 44
  - **DGColumnMassAdvection** 49
- **DGAnisotropicDiffusion** 42
  - **DGColumnHeatDispersion** 46
  - **DGColumnMassDispersion** 51

FINCH_DATA 82

GCR_DATA 114

GMRESLP_DATA 118

GMRESR_DATA 120

GMRESRP_DATA 123

GPAST_DATA 127

GSTA_DATA 129
InitialCondition
ColumnTemperatureIC 35
ConcentrationIC 36
TotalPressureIC 189
IntegratedBC
DGFluxBC 61
DGColumnWallHeatFluxBC 54
DGHeatFluxBC 66
DGMassFluxBC 73
DGFluxLimitedBC 63
DGColumnWallHeatFluxLimitedBC 57
DGHeatFluxLimitedBC 70
DGMassFluxLimitedBC 77
Kernel
AdsorptionHeatAccumulation 12
AdsorptionMassTransfer 14
BedWallHeatTransfer 26
GAdvection 101
GColumnHeatAdvection 105
GColumnMassAdvection 109
GAnisotropicDiffusion 103
GColumnHeatDispersion 107
GColumnMassDispersion 111
LinearDrivingForce 132
CoupledLDF 37
WallAmbientHeatTransfer 191
KMS_DATA 129
MAGPIE_DATA 137
Material
AdsorbentProperties 10
BedProperties 22
FlowProperties 96
MagpieAdsorbateProperties 139

Matrix< T > 143

Matrix< double > 143

MIXED_GAS 154
MooseApp

DgospreyApp 81

mSPD_DATA 157

NUM_JAC_DATA 158

OPTRANS_DATA 159

PCG_DATA 159

PICARD_DATA 162

PJFNK_DATA 164

PURE_GAS 169

SCOPSOWL_DATA 171

SCOPSOWL_PARAM_DATA 177

SKUA_DATA 180

SKUA_PARAM 184

SYSTEM_DATA 185
TimeDerivative

BedHeatAccumulation 19

BedMassAccumulation 21

WallHeatAccumulation 192

3 Class Index

3.1 Class List

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

AdsorbentProperties
AdsorbentProperties class object inherits from Material object 10

AdsorptionHeatAccumulation
AdsorptionHeatAccumulation class object inherits from Kernel object 12

AdsorptionMassTransfer
AdsorptionMassTransfer class object inherits from Kernel object 14

ARNOLDI_DATA
Data structure for the construction of the Krylov subspaces for a linear system 16

BACKTRACK_DATA
Data structure for the implementation of Backtracking Linesearch 18

BedHeatAccumulation
BedHeatAccumulation class object inherits from TimeDerivative object 19

BedMassAccumulation
BedMassAccumulation class object inherits from TimeDerivative object 21

BedProperties
BedProperties class object inherits from Material object 22

BedWallHeatTransfer
BedWallHeatTransfer class object inherits from Kernel object 26

BiCGSTAB_DATA
Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems 27

CGS_DATA
Data structure for the implementation of the CGS algorithm for non-symmetric linear systems 31

ColumnTemperatureIC
ColumnTemperatureIC class object inherits from InitialCondition object 35

ConcentrationIC
ConcentrationIC class object inherits from InitialCondition object 36

CoupledLDF
CoupledLDF class object inherits from LinearDrivingForce object 37

DGAdvection
DGAdvection class object inherits from DGKernel object 40

DGAnisotropicDiffusion
DGAnisotropicDiffusion class object inherits from DGKernel object 42

DGColumnHeatAdvection
DGColumnHeatAdvection class object inherits from DGAdvection object 44

DGColumnHeatDispersion
DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object 46

DGColumnMassAdvection
DGColumnMassAdvection class object inherits from DGAdvection object 49

DGColumnMassDispersion
DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object 51

DGColumnWallHeatFluxBC
DGColumnWallHeatFluxBC class object inherits from DGFluxBC object 54

DGColumnWallHeatFluxLimitedBC
DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object 57

DGFluxBC
DGFluxBC class object inherits from IntegratedBC object 61

DGFluxLimitedBC
DGFluxLimitedBC class object inherits from IntegratedBC object 63

DGHeatFluxBC
DGHeatFluxBC class object inherits from DGFluxBC object 66

DGHeatFluxLimitedBC
DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object 70

DGMassFluxBC
DGMassFluxBC class object inherits from DGFluxBC object 73

DGMassFluxLimitedBC
DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object 77

DgospreyApp
DgospreyApp inherits from the MooseApp object 81

FINCH_DATA
Data structure for the FINCH object 82

FlowProperties
FlowProperties class object inherits from Material object 96

GAdvection
GAdvection class object inherits from Kernel object 101

GAnisotropicDiffusion
GAnisotropicDiffusion class object inherits from Kernel object 103

GColumnHeatAdvection
GColumnHeatAdvection class object inherits from GAdvection object 105

GColumnHeatDispersion
GColumnHeatDispersion class object inherits from GAnisotropicDiffusion object 107

GColumnMassAdvection
GColumnMassAdvection class object inherits from GAdvection object 109

GColumnMassDispersion
GColumnMassDispersion class object inherits from GAnisotropicDiffusion object 111

GCR_DATA
Data structure for the implementation of the GCR algorithm for non-symmetric linear systems 114

GMRESLP_DATA
Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning 118

GMRESR_DATA
Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMR-ESR) 120

GMRESRP_DATA
Data structure for the Restarted GMRES algorithm with Right Preconditioning 123

GPAST_DATA
GPAST Data Structure 127

GSTA_DATA
GSTA Data Structure 129

KMS_DATA
Data structure for the implemenation of the Krylov Multi-Space (KMS) Method 129

LinearDrivingForce
LinearDrivingForce class object inherits from Kernel object 132

MAGPIE_Adsorption
Magpie Adsorption class inherits from AuxKernel 134

MAGPIE_AdsorptionHeat
Magpie Adsorption Heat class inherits from AuxKernel 135

MAGPIE_DATA
MAGPIE Data Structure 137

MAGPIE_Perturbation
Magpie Perturbation class inherits from AuxKernel 138

MagpieAdsorbateProperties
MagpieAdsorbateProperties class object inherits from Material object 139

Matrix< T >
Templated C++ Matrix Class Object (click Matrix to go to function definitions) 143

MIXED_GAS
Data structure holding information necessary for computing mixed gas properties 154

mSPD_DATA
MSPD Data Structure 157

NUM_JAC_DATA
Data structure to form a numerical jacobian matrix with finite differences 158

OPTRANS_DATA
Data structure for implementation of linear operator transposition 159

PCG_DATA
Data structure for implementation of the PCG algorithms for symmetric linear systems 159

PICARD_DATA
Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems 162

PJFNK_DATA
Data structure for the implementation of the PJFNK algorithm for non-linear systems 164

PURE_GAS
Data structure holding all the parameters for each pure gas spieces 169

SCOPSOWL_DATA
Primary data structure for SCOPSOWL simulations 171

SCOPSOWL_PARAM_DATA
Data structure for the species' parameters in SCOPSOWL 177

SKUA_DATA
Data structure for all simulation information in SKUA 180

SKUA_PARAM
Data structure for species' parameters in SKUA 184

SYSTEM_DATA
System Data Structure 185

TotalColumnPressure
Total Column Pressure class inherits from AuxKernel 188

TotalPressureIC
TotalPressureIC class object inherits from InitialCondition object 189

WallAmbientHeatTransfer
WallAmbientHeatTransfer class object inherits from Kernel object 191

WallHeatAccumulation
WallHeatAccumulation class object inherits from TimeDerivative object 192

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

AdsorbentProperties.h
Material Properties kernel that will setup and hold all information associated with the adsorbent 194

AdsorptionHeatAccumulation.h
Standard kernel for the heat of adsorption and its effect on the system temperature 195

AdsorptionMassTransfer.h
Standard kernel for the transfer of mass via adsorption 196

BedHeatAccumulation.h
Time Derivative kernel for the accumulation of heat in a fixed-bed column 197

BedMassAccumulation.h
Time Derivative kernel for the accumulation of mass of a species in a fixed-bed column 198

BedProperties.h 199

BedWallHeatTransfer.h
Standard kernel for the transfer of heat from the fixed-bed to the column wall 199

ColumnTemperatureIC.h
Initial Condition kernel for initial temperature in a fixed-bed column 200

ConcentrationIC.h
Initial Condition kernel for initial concentration of a species in a fixed-bed column 201

CoupledLDF.h
Advanced kernel for a cross coupled linear driving force mechanism 202

DGAdvection.h
Discontinous Galerkin kernel for advection 203

DGAnisotropicDiffusion.h
Discontinous Galerkin kernel for anisotropic diffusion 204

DGColumnHeatAdvection.h
Discontinous Galerkin kernel for energy advection in a fixed-bed column 205

DGColumnHeatDispersion.h
Discontinous Galerkin kernel for energy dispersion in a fixed-bed column 206

DGColumnMassAdvection.h
Discontinous Galerkin kernel for mass advection in a fixed-bed column 208

DGColumnMassDispersion.h
Discontinous Galerkin kernel for mass dispersion in a fixed-bed column 209

DGColumnWallHeatFluxBC.h
Boundary Condition kernel for the heat flux across the wall of the fixed-bed column 210

DGColumnWallHeatFluxLimitedBC.h
Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall 211

DGFluxBC.h
Boundary Condition kernel for the flux across a boundary of the domain 211

DGFluxLimitedBC.h
Boundary Condition kernel to mimic a Dirichlet BC for DG methods 212

DGHeatFluxBC.h
Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column 213

DGHeatFluxLimitedBC.h
Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet 214

DGMassFluxBC.h
Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column 215

DGMassFluxLimitedBC.h
Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet 216

DgospreyApp.h
Registration object for creating a registering DGOSPREY kernels 217

base/DgospreyRevision.h 218

DgospreyRevision.h 218

egret.h
Estimation of Gas-phase pRopErTies 218

error.h
All error types are defined here 222

finch.h
Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme 225

flock.h
FundamentaL Off-gas Collection of Kernels 232

FlowProperties.h
Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics 232

GAdvection.h
Kernel for use with the corresponding DGAdvection object 234

GAnisotropicDiffusion.h
Kernel for use with the corresponding DGAnisotropicDiffusion object 234

GColumnHeatAdvection.h
Kernel for use with the corresponding DGColumnHeatAdvection object 235

GColumnHeatDispersion.h
Kernel for use with the corresponding DGColumnHeatDispersion object 236

GColumnMassAdvection.h
Kernel for use with the corresponding DGColumnMassAdvection object 237

GColumnMassDispersion.h
Kernel for use with the corresponding DGColumnMassDispersion object 238

lark.h
Linear Algebra Residual Kernels 239

LinearDrivingForce.h
Standard kernel for a generic coupled linear driving force mechanism 253

macaw.h
MAtrix CAlculation Workspace 254

magpie.h
Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria 255

MAGPIE_Adsorption.h
Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system 263

MAGPIE_AdsorptionHeat.h
Auxillary kernel to calculate heat of adsorption of a particular gas species in the system 264

MAGPIE_Perturbation.h
Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system 265

MagpieAdsorbateProperties.h
Material Properties kernel that will setup and hold all information associated with MAGPIE simulations 266

scopsowl.h
Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems 267

skua.h
Surface Kinetics for Uptake by Adsorption 273

TotalColumnPressure.h
Auxillary kernel to calculate total column pressure based on temperature and concentrations 279

TotalPressureIC.h
Initial Condition kernel for initial temperature in a fixed-bed column 280

WallAmbientHeatTransfer.h
Standard kernel for the transfer of heat from the column wall to the ambient air 281

WallHeatAccumulation.h
Time Derivative kernel for the accumulation of heat in a walls of the column 282

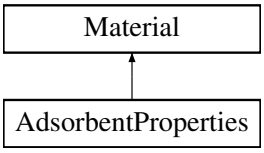
5 Class Documentation

5.1 AdsorbentProperties Class Reference

AdsorbentProperties class object inherits from Material object.

`#include <AdsorbentProperties.h>`

Inheritance diagram for AdsorbentProperties:



Public Member Functions

- AdsorbentProperties (const InputParameters &parameters)
  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual void computeQpProperties ()
  Required function override for Material objects in MOOSE.

Private Attributes

- Real _binder_fraction
  Binder fraction in the biporous adsorbent pellet (0 means no binder material)
- Real _binder_porosity
  Macro-porosity of the binder material in the adsorbent pellet.
- Real _crystal_radius
  Nominal radius of the adsorbent crystals suspended in the binder (um)
- Real _pellet_diameter
  Nominal diameter of the adsorbent pellets in the system (cm)
- Real _macropore_radius
  Nominal size of the macro-pores in the pellet (cm)
- Real _rhos
  Density of the adsorbent pellet (g/cm^3)
- Real _hs
  Heat capacity of the adsorbent pellet (J/g/K)
- MaterialProperty< Real > & _pellet_density
  MaterialProperty for the pellet density (g/cm^3)
- MaterialProperty< Real > & _pellet_heat_capacity
  MaterialProperty for the pellet heat capacity (J/g/K)
- VariableValue & _temperature
  Reference to the coupled column temperature.
- std::vector< unsigned int > _index
  List of indices for the coupled gases.
- std::vector< VariableValue * > _gas_conc
  Pointer list for the coupled gases.

5.1.1 Detailed Description

AdsorbentProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up the structural information about adsorbent in the system that will be used when determining flow properties, as well as some kinetic properties for adsorption dynamics.

Definition at line 57 of file AdsorbentProperties.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 AdsorbentProperties::AdsorbentProperties (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.1.3 Member Function Documentation

5.1.3.1 virtual void AdsorbentProperties::computeQpProperties ( ) [protected], [virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

5.1.4 Member Data Documentation

5.1.4.1 Real AdsorbentProperties::_binder_fraction [private]

Binder fraction in the biporous adsorbent pellet (0 means no binder material)

Definition at line 69 of file AdsorbentProperties.h.

5.1.4.2 Real AdsorbentProperties::_binder_porosity [private]

Macro-porosity of the binder material in the adsorbent pellet.

Definition at line 70 of file AdsorbentProperties.h.

5.1.4.3 Real AdsorbentProperties::_crystal_radius [private]

Nominal radius of the adsorbent crystals suspended in the binder (um)

Definition at line 71 of file AdsorbentProperties.h.

5.1.4.4 std::vector<VariableValue *> AdsorbentProperties::_gas_conc [private]

Pointer list for the coupled gases.

Definition at line 82 of file AdsorbentProperties.h.

5.1.4.5 Real AdsorbentProperties::_hs [private]

Heat capacity of the adsorbent pellet (J/g/K)

Definition at line 75 of file AdsorbentProperties.h.

5.1.4.6 std::vector<unsigned int> AdsorbentProperties::_index [private]

List of indices for the coupled gases.

Definition at line 81 of file AdsorbentProperties.h.

5.1.4.7 Real AdsorbentProperties::_macropore_radius [private]

Nominal size of the macro-pores in the pellet (cm)

Definition at line 73 of file AdsorbentProperties.h.

5.1.4.8 MaterialProperty<Real>& AdsorbentProperties::_pellet_density [private]

MaterialProperty for the pellet density (g/cm^3)

Definition at line 77 of file AdsorbentProperties.h.

5.1.4.9 Real AdsorbentProperties::_pellet_diameter [private]

Nominal diameter of the adsorbent pellets in the system (cm)

Definition at line 72 of file AdsorbentProperties.h.

5.1.4.10 MaterialProperty<Real>& AdsorbentProperties::_pellet_heat_capacity [private]

MaterialProperty for the pellet heat capacity (J/g/K)

Definition at line 78 of file AdsorbentProperties.h.

5.1.4.11 Real AdsorbentProperties::_rhos [private]

Density of the adsorbent pellet (g/cm^3)

Definition at line 74 of file AdsorbentProperties.h.

5.1.4.12 VariableValue& AdsorbentProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 80 of file AdsorbentProperties.h.

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

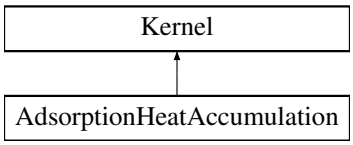
- AdsorbentProperties.h

5.2 AdsorptionHeatAccumulation Class Reference

AdsorptionHeatAccumulation class object inherits from Kernel object.

`#include <AdsorptionHeatAccumulation.h>`

Inheritance diagram for AdsorptionHeatAccumulation:



Public Member Functions

- AdsorptionHeatAccumulation (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- const MaterialProperty< Real > & _porosity

  Reference to the bed bulk porosity material property.

- const MaterialProperty< Real > & _pellet_density

  Reference to the pellet density material property.
- std::vector< VariableValue * > _solid_heat

  Pointer list to the coupled heats of adsorption at the current time.
- std::vector< VariableValue * > _solid_heat_old

  Pointer list to the coupled heats of adsorption at the previous time.

5.2.1 Detailed Description

AdsorptionHeatAccumulation class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel interfaces the material properties for the bulk bed porosity and the pellet density, as well as coupling with the heat of adsorption as it changes in time, in order to form a residuals and Jacobians for the gas temperature variable.

Definition at line 55 of file AdsorptionHeatAccumulation.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 AdsorptionHeatAccumulation::AdsorptionHeatAccumulation (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.2.3 Member Function Documentation

5.2.3.1 virtual Real AdsorptionHeatAccumulation::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.2.3.2 virtual Real AdsorptionHeatAccumulation::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.2.4 Member Data Documentation

5.2.4.1 const MaterialProperty<Real>& AdsorptionHeatAccumulation::_pellet_density [private]

Reference to the pellet density material property.

Definition at line 73 of file AdsorptionHeatAccumulation.h.

5.2.4.2 const MaterialProperty<Real>& AdsorptionHeatAccumulation::_porosity [private]

Reference to the bed bulk porosity material property.

Definition at line 72 of file AdsorptionHeatAccumulation.h.

5.2.4.3 std::vector<VariableValue *> AdsorptionHeatAccumulation::_solid_heat [private]

Pointer list to the coupled heats of adsorption at the current time.

Definition at line 74 of file AdsorptionHeatAccumulation.h.

5.2.4.4 std::vector<VariableValue *> AdsorptionHeatAccumulation::_solid_heat_old [private]

Pointer list to the coupled heats of adsorption at the previous time.

Definition at line 75 of file AdsorptionHeatAccumulation.h.

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

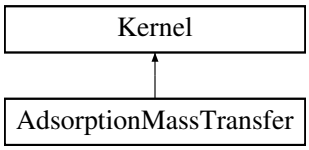
- AdsorptionHeatAccumulation.h

5.3 AdsorptionMassTransfer Class Reference

AdsorptionMassTransfer class object inherits from Kernel object.

`#include <AdsorptionMassTransfer.h>`

Inheritance diagram for AdsorptionMassTransfer:



Public Member Functions

- AdsorptionMassTransfer (const InputParameters &parameters)
  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()
  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()
  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- const MaterialProperty< Real > & _porosity
  Reference to the bed bulk porosity material property.
- const MaterialProperty< Real > & _pellet_density
  Reference to the pellet density material property.
- VariableValue & _solid
  Pointer to coupled adsorption at the current time.
- VariableValue & _solid_old
  Pointer to coupled adsorption at the previous time.

5.3.1 Detailed Description

AdsorptionMassTransfer class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel interfaces the material properties for the bulk bed porosity and the pellet density, as well as coupling with adsorption as it changes in time, in order to form a residuals and Jacobians for the gas concentration variable.

Definition at line 54 of file AdsorptionMassTransfer.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 AdsorptionMassTransfer::AdsorptionMassTransfer (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.3.3 Member Function Documentation

5.3.3.1 virtual Real AdsorptionMassTransfer::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.3.3.2 virtual Real AdsorptionMassTransfer::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.3.4 Member Data Documentation

5.3.4.1 const MaterialProperty<Real>& AdsorptionMassTransfer::_pellet_density [private]

Reference to the pellet density material property.

Definition at line 72 of file AdsorptionMassTransfer.h.

5.3.4.2 const MaterialProperty<Real>& AdsorptionMassTransfer::_porosity [private]

Reference to the bed bulk porosity material property.

Definition at line 71 of file AdsorptionMassTransfer.h.

5.3.4.3 VariableValue& AdsorptionMassTransfer::_solid [private]

Pointer to coupled adsorption at the current time.

Definition at line 73 of file AdsorptionMassTransfer.h.

5.3.4.4 VariableValue& AdsorptionMassTransfer::_solid_old [private]

Pointer to coupled adsorption at the previous time.

Definition at line 74 of file AdsorptionMassTransfer.h.

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

- AdsorptionMassTransfer.h

5.4 ARNOLDI_DATA Struct Reference

Data structure for the construction of the Krylov subspaces for a linear system.

`#include <lark.h>`

Public Attributes

- int k

Desired size of the Krylov subspace.

- int iter

  Actual size of the Krylov subspace.

- double beta

  Normalization parameter.

- double hp1

  Additional row element of H (separate storage for holding)

- bool Output = true

  True = print messages to console.

- std::vector< Matrix< double > > Vk

  (N) x (k) orthonormal vector basis stored as a vector of column matrices

- Matrix< double > Hkp1

  (k+1) x (k) upper Hessenberg matrix

- Matrix< double > yk

  (k) x (1) vector search direction

- Matrix< double > e1

  (k) x (1) orthonormal vector with 1 in first position

- Matrix< double > w

  (N) x (1) interim result of the matrix_vector multiplication

- Matrix< double > v

  (N) x (1) holding cell for the column entries of Vk and other interims

- Matrix< double > sum

  (N) x (1) running sum of subspace vectors for use in altering w

5.4.1 Detailed Description

Data structure for the construction of the Krylov subspaces for a linear system.

C-style object used in conjunction with the Arnoldi algorithm to construct an orthonormal basis and upper Hessenberg representation of a given linear operator. This is used to solve a linear system both iteratively (i.e., in conjunction with GMRESLP) and directly (i.e., in conjunction with FOM). Alternatively, you can just store the factorized components for later use in another routine.

Definition at line 120 of file lark.h.

5.4.2 Member Data Documentation

5.4.2.1 double ARNOLDI_DATA::beta

Normalization parameter.

Definition at line 125 of file lark.h.

5.4.2.2 Matrix<double> ARNOLDI_DATA::e1

(k) x (1) orthonormal vector with 1 in first position

Definition at line 133 of file lark.h.

5.4.2.3 Matrix<double> ARNOLDI_DATA::Hkp1

(k+1) x (k) upper Hessenberg matrix

Definition at line 131 of file lark.h.

5.4.2.4 double ARNOLDI_DATA::hp1

Additional row element of H (separate storage for holding)

Definition at line 126 of file lark.h.

5.4.2.5 int ARNOLDI_DATA::iter

Actual size of the Krylov subspace.

Definition at line 123 of file lark.h.

5.4.2.6 int ARNOLDI_DATA::k

Desired size of the Krylov subspace.

Definition at line 122 of file lark.h.

5.4.2.7 bool ARNOLDI_DATA::Output = true

True = print messages to console.

Definition at line 128 of file lark.h.

5.4.2.8 Matrix<double> ARNOLDI_DATA::sum

(N) x (1) running sum of subspace vectors for use in altering w

Definition at line 136 of file lark.h.

5.4.2.9 Matrix<double> ARNOLDI_DATA::v

(N) x (1) holding cell for the column entries of Vk and other interims

Definition at line 135 of file lark.h.

5.4.2.10 std::vector< Matrix<double> > ARNOLDI_DATA::Vk

(N) x (k) orthonormal vector basis stored as a vector of column matrices

Definition at line 130 of file lark.h.

5.4.2.11 Matrix<double> ARNOLDI_DATA::w

(N) x (1) interim result of the matrix_vector multiplication

Definition at line 134 of file lark.h.

5.4.2.12 Matrix<double> ARNOLDI_DATA::yk

(k) x (1) vector search direction

Definition at line 132 of file lark.h.

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

- lark.h

5.5 BACKTRACK_DATA Struct Reference

Data structure for the implementation of Backtracking Linesearch.

`#include <lark.h>`

Public Attributes

- int fun_call = 0

  Number of function calls made during line search.
- double alpha = 1e-4

  Scaling parameter for determination of search step size.
- double rho = 0.1

  Scaling parameter for to change step size by.
- double lambdaMin =DBL_EPSILON

  Smallest allowable step length.
- double normFkp1

  New residual norm of the Newton step.
- bool constRho = false

  True = use a constant value for rho.
- Matrix< double > Fk

  Old residual vector of the Newton step.
- Matrix< double > xk

  Old solution vector of the Newton step.

5.5.1 Detailed Description

Data structure for the implementation of Backtracking Linesearch.

C-style object used in conjunction with the Backtracking Linesearch algorithm to smooth out convergence of Netwon based iterative methods for non-linear systems of equations. The actual algorithm has been separated from the interior of the Newton method so that it can be included in any future Newton based iterative methods being developed.

Definition at line 474 of file lark.h.

5.5.2 Member Data Documentation

5.5.2.1 double BACKTRACK_DATA::alpha = 1e-4

Scaling parameter for determination of search step size.

Definition at line 477 of file lark.h.

5.5.2.2 bool BACKTRACK_DATA::constRho = false

True = use a constant value for rho.

Definition at line 482 of file lark.h.

5.5.2.3 Matrix<double> BACKTRACK_DATA::Fk

Old residual vector of the Newton step.

Definition at line 484 of file lark.h.

5.5.2.4 int BACKTRACK_DATA::fun_call = 0

Number of function calls made during line search.

Definition at line 476 of file lark.h.

5.5.2.5 double BACKTRACK_DATA::lambdaMin =DBL_EPSILON

Smallest allowable step length.

Definition at line 479 of file lark.h.

5.5.2.6 double BACKTRACK_DATA::normFkp1

New residual norm of the Newton step.

Definition at line 480 of file lark.h.

5.5.2.7 double BACKTRACK_DATA::rho = 0.1

Scaling parameter for to change step size by.

Definition at line 478 of file lark.h.

5.5.2.8 Matrix<double> BACKTRACK_DATA::xk

Old solution vector of the Newton step.

Definition at line 485 of file lark.h.

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

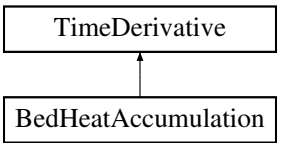
- lark.h

5.6 BedHeatAccumulation Class Reference

BedHeatAccumulation class object inherits from TimeDerivative object.

`#include <BedHeatAccumulation.h>`

Inheritance diagram for BedHeatAccumulation:



Public Member Functions

- BedHeatAccumulation (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- const MaterialProperty< Real > & _heat_retardation

  Reference to the heat retardation material property.

5.6.1 Detailed Description

BedHeatAccumulation class object inherits from TimeDerivative object.

This class object inherits from the TimeDerivative object. All public and protected members of this class are required function overrides. The kernel interfaces with the heat retardation coefficient calculated in a materials property file and calls the standard TimeDerivative functions while appending the retardation coefficient to those values.

Definition at line 54 of file BedHeatAccumulation.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 BedHeatAccumulation::BedHeatAccumulation (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.6.3 Member Function Documentation

5.6.3.1 virtual Real BedHeatAccumulation::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.6.3.2 virtual Real BedHeatAccumulation::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.6.4 Member Data Documentation

5.6.4.1 const MaterialProperty<Real>& BedHeatAccumulation::_heat_retardation [private]

Reference to the heat retardation material property.

Definition at line 71 of file BedHeatAccumulation.h.

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

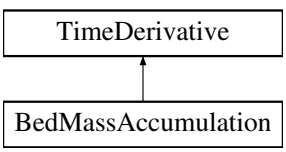
- BedHeatAccumulation.h

5.7 BedMassAccumulation Class Reference

BedMassAccumulation class object inherits from TimeDerivative object.

`#include <BedMassAccumulation.h>`

Inheritance diagram for BedMassAccumulation:



Public Member Functions

- BedMassAccumulation (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- int _index

  Index of the species of interest for the mass accumulation.
- const MaterialProperty
  < std::vector< Real > > & _retardation

  Reference to the mass retardation coefficient material property.

5.7.1 Detailed Description

BedMassAccumulation class object inherits from TimeDerivative object.

This class object inherits from the TimeDerivative object. All public and protected members of this class are required function overrides. The kernel interfaces with the mass retardation coefficient calculated in a materials property file and calls the standard TimeDerivative functions while appending the retardation coefficient to those values.

Definition at line 54 of file BedMassAccumulation.h.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 BedMassAccumulation::BedMassAccumulation (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.7.3 Member Function Documentation

5.7.3.1 virtual Real BedMassAccumulation::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.7.3.2 virtual Real BedMassAccumulation::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.7.4 Member Data Documentation

5.7.4.1 int BedMassAccumulation::_index [private]

Index of the species of interest for the mass accumulation.

Definition at line 71 of file BedMassAccumulation.h.

5.7.4.2 const MaterialProperty<std::vector<Real> >& BedMassAccumulation::_retardation [private]

Reference to the mass retardation coefficient material property.

Definition at line 72 of file BedMassAccumulation.h.

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

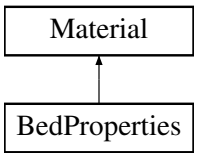
- BedMassAccumulation.h

5.8 BedProperties Class Reference

BedProperties class object inherits from Material object.

`#include <BedProperties.h>`

Inheritance diagram for BedProperties:



Public Member Functions

- BedProperties (const InputParameters &parameters)
  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual void computeQpProperties ()
  Required function override for Material objects in MOOSE.

Private Attributes

- Real _length
  Bed length (cm)
- Real _din
  Column inner diameter (cm)
- Real _dout
  Column outer diameter (cm)
- Real _eb
  Bulk porosity of the bed.
- Real _Kz
  Axial thermal conductivity of the bed (J/hr/cm/K)
- Real _rhow

Density of the column wall (g/cm^3)

- Real _hw

  Heat capacity of the column wall (J/g/K)
- Real _Uw

  Bed-Wall heat transfer coefficient (J/hr/cm^2/K)
- Real _Ua

  External-Wall heat transfer coefficient (J/hr/cm^2/K)
- MaterialProperty< Real > & _inner_dia

  MaterialProperty for column inner diameter.
- MaterialProperty< Real > & _outer_dia

  MaterialProperty for column outer diameter.
- MaterialProperty< Real > & _porosity

  MaterialProperty for bulk porosity of the bed.
- MaterialProperty< Real > & _conductivity

  MaterialProperty for thermal conductivity of the bed.
- MaterialProperty< Real > & _wall_density

  MaterialProperty for column wall density.
- MaterialProperty< Real > & _wall_heat_capacity

  MaterialProperty for column wall heat capacity.
- MaterialProperty< Real > & _bed_wall_transfer_coeff

  MaterialProperty for bed-wall heat transfer coefficient.
- MaterialProperty< Real > & _wall_exterior_transfer_coeff

  MaterialProperty for exterior-wall heat transfer coefficient.
- VariableValue & _temperature

  Reference to the coupled column temperature.
- std::vector< unsigned int > _index

  List of indices for the species in the system.
- std::vector< VariableValue * > _gas_conc

  Pointer list of the gas species concentrations.

5.8.1 Detailed Description

BedProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up the parameters of the fixed-bed column. Those parameters include: length, diameter, thermal conductivity, heat transfer coefficients, bulk porosity, etc.

Definition at line 55 of file BedProperties.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 BedProperties::BedProperties (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.8.3 Member Function Documentation

5.8.3.1 virtual void BedProperties::computeQpProperties ( ) [protected], [virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

5.8.4 Member Data Documentation

5.8.4.1 MaterialProperty<Real>& BedProperties::_bed_wall_transfer_coeff [private]

MaterialProperty for bed-wall heat transfer coefficient.

Definition at line 83 of file BedProperties.h.

5.8.4.2 MaterialProperty<Real>& BedProperties::_conductivity [private]

MaterialProperty for thermal conductivity of the bed.

Definition at line 80 of file BedProperties.h.

5.8.4.3 Real BedProperties::_din [private]

Column inner diameter (cm)

Definition at line 68 of file BedProperties.h.

5.8.4.4 Real BedProperties::_dout [private]

Column outer diameter (cm)

Definition at line 69 of file BedProperties.h.

5.8.4.5 Real BedProperties::_eb [private]

Bulk porosity of the bed.

Definition at line 70 of file BedProperties.h.

5.8.4.6 std::vector<VariableValue *> BedProperties::_gas_conc [private]

Pointer list of the gas species concentrations.

Definition at line 88 of file BedProperties.h.

5.8.4.7 Real BedProperties::_hw [private]

Heat capacity of the column wall (J/g/K)

Definition at line 73 of file BedProperties.h.

5.8.4.8 std::vector<unsigned int> BedProperties::_index [private]

List of indices for the species in the system.

Definition at line 87 of file BedProperties.h.

5.8.4.9 MaterialProperty<Real>& BedProperties::_inner_dia [private]

MaterialProperty for column inner diameter.

Definition at line 77 of file BedProperties.h.

5.8.4.10 Real BedProperties::_Kz [private]

Axial thermal conductivity of the bed (J/hr/cm/K)

Definition at line 71 of file BedProperties.h.

5.8.4.11 Real BedProperties::_length [private]

Bed length (cm)

Definition at line 67 of file BedProperties.h.

5.8.4.12 MaterialProperty<Real>& BedProperties::_outer_dia [private]

MaterialProperty for column outer diameter.

Definition at line 78 of file BedProperties.h.

5.8.4.13 MaterialProperty<Real>& BedProperties::_porosity [private]

MaterialProperty for bulk porosity of the bed.

Definition at line 79 of file BedProperties.h.

5.8.4.14 Real BedProperties::_rhow [private]

Density of the column wall (g/cm^3)

Definition at line 72 of file BedProperties.h.

5.8.4.15 VariableValue& BedProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 86 of file BedProperties.h.

5.8.4.16 Real BedProperties::_Ua [private]

External-Wall heat transfer coefficient (J/hr/cm^2/K)

Definition at line 75 of file BedProperties.h.

5.8.4.17 Real BedProperties::_Uw [private]

Bed-Wall heat transfer coefficient (J/hr/cm^2/K)

Definition at line 74 of file BedProperties.h.

5.8.4.18 MaterialProperty<Real>& BedProperties::_wall_density [private]

MaterialProperty for column wall density.

Definition at line 81 of file BedProperties.h.

5.8.4.19 MaterialProperty<Real>& BedProperties::_wall_exterior_transfer_coeff [private]

MaterialProperty for exterior-wall heat transfer coefficient.

Definition at line 84 of file BedProperties.h.

5.8.4.20 MaterialProperty<Real>& BedProperties::_wall_heat_capacity [private]

MaterialProperty for column wall heat capacity.

Definition at line 82 of file BedProperties.h.

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

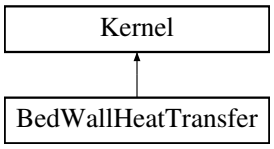
- BedProperties.h

5.9 BedWallHeatTransfer Class Reference

BedWallHeatTransfer class object inherits from Kernel object.

`#include <BedWallHeatTransfer.h>`

Inheritance diagram for BedWallHeatTransfer:



Public Member Functions

- BedWallHeatTransfer (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- const MaterialProperty< Real > & _bed_wall_transfer_coeff

  Reference to the bed-wall heat transfer material property.
- const MaterialProperty< Real > & _inner_dia

  Reference to the wall inner diameter material property.
- const MaterialProperty< Real > & _outer_dia

  Reference to the wall outer diameter material property.
- VariableValue & _column_temp

  Reference to the gas temperature coupled non-linear variable.

5.9.1 Detailed Description

BedWallHeatTransfer class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel interfaces the material properties for the size of the column, as well as the heat transfer coefficient for the exchange of energy from the gas to the wall, in order to form a residuals and Jacobians for the wall temperature variable.

Definition at line 56 of file BedWallHeatTransfer.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 BedWallHeatTransfer::BedWallHeatTransfer (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.9.3 Member Function Documentation

5.9.3.1 virtual Real BedWallHeatTransfer::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.9.3.2 virtual Real BedWallHeatTransfer::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.9.4 Member Data Documentation

5.9.4.1 const MaterialProperty<Real>& BedWallHeatTransfer::_bed_wall_transfer_coeff [private]

Reference to the bed-wall heat transfer material property.

Definition at line 73 of file BedWallHeatTransfer.h.

5.9.4.2 VariableValue& BedWallHeatTransfer::_column_temp [private]

Reference to the gas temperature coupled non-linear variable.

Definition at line 77 of file BedWallHeatTransfer.h.

5.9.4.3 const MaterialProperty<Real>& BedWallHeatTransfer::_inner_dia [private]

Reference to the wall inner diameter material property.

Definition at line 74 of file BedWallHeatTransfer.h.

5.9.4.4 const MaterialProperty<Real>& BedWallHeatTransfer::_outer_dia [private]

Reference to the wall outer diameter material property.

Definition at line 75 of file BedWallHeatTransfer.h.

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

- BedWallHeatTransfer.h

5.10 BiCGSTAB_DATA Struct Reference

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

`#include <lark.h>`

Public Attributes

- int maxit = 0

  Maximum allowable iterations - default = min(2∗vector_size,1000)
- int iter = 0

  Actual number of iterations.
- bool breakdown

  Boolean to determine if the method broke down.
- double alpha

  Step size parameter for next solution.
- double beta

  Step size parameter for search direction.
- double rho

Scaling parameter for alpha and beta.

- double rho_old

  Previous scaling parameter for alpha and beta.
- double omega

  Scaling parameter and additional step length.
- double omega_old

  Previous scaling parameter and step length.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm.
- double relres

  Relative residual norm.
- double relres_base

  Initial residual norm.
- double bestres

  Best found residual norm.
- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual vector for the linear system.
- Matrix< double > r0

  Initial residual vector.
- Matrix< double > v

  Search direction for p.
- Matrix< double > p

  Search direction for updating.
- Matrix< double > y

  Preconditioned search direction.
- Matrix< double > s

  Residual updating vector.
- Matrix< double > z

  Preconditioned residual updating vector.
- Matrix< double > t

  Search direction for resdidual updates.

5.10.1 Detailed Description

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

C-style object used in conjunction with the Bi-Conjugate Gradient STABalized (BiCGSTAB) algorithm to solve a linear system of equations. This algorithm is generally more efficient than any GMRES or GCR variant, but may not always reduce the residual at each step. However, if used with preconditioning, then this algorithm is very efficient, especially when used for solving grid-based linear systems.

Definition at line 249 of file lark.h.

5.10.2 Member Data Documentation

5.10.2.1 double BiCGSTAB_DATA::alpha

Step size parameter for next solution.

Definition at line 255 of file lark.h.

5.10.2.2 double BiCGSTAB_DATA::bestres

Best found residual norm.

Definition at line 266 of file lark.h.

5.10.2.3 Matrix<double> BiCGSTAB_DATA::bestx

Best found solution to the linear system.

Definition at line 271 of file lark.h.

5.10.2.4 double BiCGSTAB_DATA::beta

Step size parameter for search direction.

Definition at line 256 of file lark.h.

5.10.2.5 bool BiCGSTAB_DATA::breakdown

Boolean to determine if the method broke down.

Definition at line 253 of file lark.h.

5.10.2.6 int BiCGSTAB_DATA::iter = 0

Actual number of iterations.

Definition at line 252 of file lark.h.

5.10.2.7 int BiCGSTAB_DATA::maxit = 0

Maximum allowable iterations - default = min(2∗vector_size,1000)

Definition at line 251 of file lark.h.

5.10.2.8 double BiCGSTAB_DATA::omega

Scaling parameter and additional step length.

Definition at line 259 of file lark.h.

5.10.2.9 double BiCGSTAB_DATA::omega_old

Previous scaling parameter and step length.

Definition at line 260 of file lark.h.

5.10.2.10 bool BiCGSTAB_DATA::Output = true

True = print messages to console.

Definition at line 268 of file lark.h.

5.10.2.11 Matrix<double> BiCGSTAB_DATA::p

Search direction for updating.

Definition at line 275 of file lark.h.

5.10.2.12 Matrix<double> BiCGSTAB_DATA::r

Residual vector for the linear system.

Definition at line 272 of file lark.h.

5.10.2.13 Matrix<double> BiCGSTAB_DATA::r0

Initial residual vector.

Definition at line 273 of file lark.h.

5.10.2.14 double BiCGSTAB_DATA::relres

Relative residual norm.

Definition at line 264 of file lark.h.

5.10.2.15 double BiCGSTAB_DATA::relres_base

Initial residual norm.

Definition at line 265 of file lark.h.

5.10.2.16 double BiCGSTAB_DATA::res

Absolute residual norm.

Definition at line 263 of file lark.h.

5.10.2.17 double BiCGSTAB_DATA::rho

Scaling parameter for alpha and beta.

Definition at line 257 of file lark.h.

5.10.2.18 double BiCGSTAB_DATA::rho_old

Previous scaling parameter for alpha and beta.

Definition at line 258 of file lark.h.

5.10.2.19 Matrix<double> BiCGSTAB_DATA::s

Residual updating vector.

Definition at line 277 of file lark.h.

5.10.2.20 Matrix<double> BiCGSTAB_DATA::t

Search direction for resdidual updates.

Definition at line 279 of file lark.h.

5.10.2.21 double BiCGSTAB_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 262 of file lark.h.

5.10.2.22 double BiCGSTAB_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 261 of file lark.h.

5.10.2.23 Matrix<double> BiCGSTAB_DATA::v

Search direction for p.

Definition at line 274 of file lark.h.

5.10.2.24 Matrix<double> BiCGSTAB_DATA::x

Current solution to the linear system.

Definition at line 270 of file lark.h.

5.10.2.25 Matrix<double> BiCGSTAB_DATA::y

Preconditioned search direction.

Definition at line 276 of file lark.h.

5.10.2.26 Matrix<double> BiCGSTAB_DATA::z

Preconditioned residual updating vector.

Definition at line 278 of file lark.h.

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

- lark.h

5.11 CGS_DATA Struct Reference

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

`#include <lark.h>`

Public Attributes

- int maxit = 0
  *Maximum allowable iterations - default = min(2*vector_size,1000)*
- int iter = 0
  Actual number of iterations.
- bool breakdown
  Boolean to determine if the method broke down.
- double alpha
  Step size parameter for next solution.
- double beta
  Step size parameter for search direction.
- double rho
  Scaling parameter for alpha and beta.
- double sigma
  Scaling parameter and additional step length.
- double tol_rel = 1e-6
  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6
  Absolution tolerance for convergence - default = 1e-6.
- double res
  Absolute residual norm.
- double relres

Relative residual norm.

- double relres_base

  Initial residual norm.

- double bestres

  Best found residual norm.

- bool Output = true

  True = print messages to console.

- Matrix< double > x

  Current solution to the linear system.

- Matrix< double > bestx

  Best found solution to the linear system.

- Matrix< double > r

  Residual vector for the linear system.

- Matrix< double > r0

  Initial residual vector.

- Matrix< double > u

  Search direction for v.

- Matrix< double > w

  Updates sigma and u.

- Matrix< double > v

  Search direction for x.

- Matrix< double > p

  Preconditioning result for w, z, and matvec for Ax.

- Matrix< double > c

  Holds the matvec result between A and p.

- Matrix< double > z

  Full search direction for x.

5.11.1 Detailed Description

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

C-style object to be used in conjunction with the Conjugate Gradient Squared (CGS) algorithm to solve linear systems of equations. This algorithm is slightly less computational work than BiCGSTAB, but is much less stable. As a result, I do not recommend using this algorithm unless you also use some form of preconditioning.

Definition at line 288 of file lark.h.

5.11.2 Member Data Documentation

5.11.2.1 double CGS_DATA::alpha

Step size parameter for next solution.

Definition at line 294 of file lark.h.

5.11.2.2 double CGS_DATA::bestres

Best found residual norm.

Definition at line 303 of file lark.h.

5.11.2.3 Matrix<double> CGS_DATA::bestx

Best found solution to the linear system.

Definition at line 308 of file lark.h.

5.11.2.4 double CGS_DATA::beta

Step size parameter for search direction.

Definition at line 295 of file lark.h.

5.11.2.5 bool CGS_DATA::breakdown

Boolean to determine if the method broke down.

Definition at line 292 of file lark.h.

5.11.2.6 Matrix<double> CGS_DATA::c

Holds the matvec result between A and p.

Definition at line 315 of file lark.h.

5.11.2.7 int CGS_DATA::iter = 0

Actual number of iterations.

Definition at line 291 of file lark.h.

5.11.2.8 int CGS_DATA::maxit = 0

Maximum allowable iterations - default = min(2∗vector_size,1000)

Definition at line 290 of file lark.h.

5.11.2.9 bool CGS_DATA::Output = true

True = print messages to console.

Definition at line 305 of file lark.h.

5.11.2.10 Matrix<double> CGS_DATA::p

Preconditioning result for w, z, and matvec for Ax.

Definition at line 314 of file lark.h.

5.11.2.11 Matrix<double> CGS_DATA::r

Residual vector for the linear system.

Definition at line 309 of file lark.h.

5.11.2.12 Matrix<double> CGS_DATA::r0

Initial residual vector.

Definition at line 310 of file lark.h.

5.11.2.13 double CGS_DATA::relres

Relative residual norm.

Definition at line 301 of file lark.h.

5.11.2.14 double CGS_DATA::relres_base

Initial residual norm.

Definition at line 302 of file lark.h.

5.11.2.15 double CGS_DATA::res

Absolute residual norm.

Definition at line 300 of file lark.h.

5.11.2.16 double CGS_DATA::rho

Scaling parameter for alpha and beta.

Definition at line 296 of file lark.h.

5.11.2.17 double CGS_DATA::sigma

Scaling parameter and additional step length.

Definition at line 297 of file lark.h.

5.11.2.18 double CGS_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 299 of file lark.h.

5.11.2.19 double CGS_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 298 of file lark.h.

5.11.2.20 Matrix<double> CGS_DATA::u

Search direction for v.

Definition at line 311 of file lark.h.

5.11.2.21 Matrix<double> CGS_DATA::v

Search direction for x.

Definition at line 313 of file lark.h.

5.11.2.22 Matrix<double> CGS_DATA::w

Updates sigma and u.

Definition at line 312 of file lark.h.

5.11.2.23 Matrix<double> CGS_DATA::x

Current solution to the linear system.

Definition at line 307 of file lark.h.

5.11.2.24 Matrix<double> CGS_DATA::z

Full search direction for x.

Definition at line 316 of file lark.h.

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

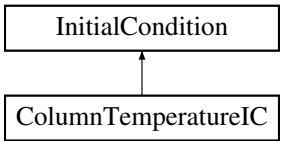
- lark.h

5.12 ColumnTemperatureIC Class Reference

ColumnTemperatureIC class object inherits from InitialCondition object.

`#include <ColumnTemperatureIC.h>`

Inheritance diagram for ColumnTemperatureIC:



Public Member Functions

- ColumnTemperatureIC (const InputParameters &parameters)

  Required constructor for objects in MOOSE.
- virtual Real value (const Point &p)

  Required function override for setting the value of the non-linear variable at a given point.

Private Attributes

- Real _TC_IC

  Initial condition value for the column temperature (K)

5.12.1 Detailed Description

ColumnTemperatureIC class object inherits from InitialCondition object.

This class object inherits from the InitialCondition object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will establish the initial conditions for column temperature as constant throughout the domain.

Note

You can have the non-linear variable vary spatially in the domain by inheriting from and or modifying this file to do so.

Definition at line 58 of file ColumnTemperatureIC.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 ColumnTemperatureIC::ColumnTemperatureIC (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.12.3 Member Function Documentation

5.12.3.1 virtual Real ColumnTemperatureIC::value (const Point & *p*) [virtual]

Required function override for setting the value of the non-linear variable at a given point.

This function passes a point p as an argument. The return value will be the value of the non-linear variable at that point. That information is used to establish the spatially varying initial condition for the given non-linear variable.

5.12.4 Member Data Documentation

5.12.4.1 Real ColumnTemperatureIC::_TC_IC [private]

Initial condition value for the column temperature (K)

Definition at line 70 of file ColumnTemperatureIC.h.

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

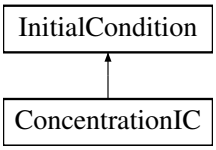
- ColumnTemperatureIC.h

5.13 ConcentrationIC Class Reference

ConcentrationIC class object inherits from InitialCondition object.

`#include <ConcentrationIC.h>`

Inheritance diagram for ConcentrationIC:



Public Member Functions

- ConcentrationIC (const InputParameters &parameters)

  Required constructor for objects in MOOSE.
- virtual Real value (const Point &p)

  Required function override for setting the value of the non-linear variable at a given point.

Private Attributes

- Real _y_IC

  Initial molefraction of the species in the gas phase.
- Real _PT_IC

  Initial total pressure in the column (kPa)
- Real _T_IC

  Initial temperature in the column (K)

5.13.1 Detailed Description

ConcentrationIC class object inherits from InitialCondition object.

This class object inherits from the InitialCondition object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will establish the initial conditions for a species' concentration as constant throughout the domain.

Note

You can have the non-linear variable vary spatially in the domain by inheriting from and or modifying this file to do so.

Definition at line 58 of file ConcentrationIC.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 ConcentrationIC::ConcentrationIC (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.13.3 Member Function Documentation

5.13.3.1 virtual Real ConcentrationIC::value (const Point & *p*) [virtual]

Required function override for setting the value of the non-linear variable at a given point.

This function passes a point p as an argument. The return value will be the value of the non-linear variable at that point. That information is used to establish the spatially varying initial condition for the given non-linear variable.

5.13.4 Member Data Documentation

5.13.4.1 Real ConcentrationIC::_PT_IC [private]

Initial total pressure in the column (kPa)

Definition at line 71 of file ConcentrationIC.h.

5.13.4.2 Real ConcentrationIC::_T_IC [private]

Initial temperature in the column (K)

Definition at line 72 of file ConcentrationIC.h.

5.13.4.3 Real ConcentrationIC::_y_IC [private]

Initial molefraction of the species in the gas phase.

Definition at line 70 of file ConcentrationIC.h.

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

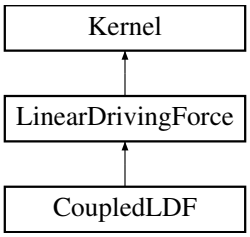
- ConcentrationIC.h

5.14 CoupledLDF Class Reference

CoupledLDF class object inherits from LinearDrivingForce object.

`#include <CoupledLDF.h>`

Inheritance diagram for CoupledLDF:



Public Member Functions

- CoupledLDF (const InputParameters &parameters)
  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- Real _drive_coef

  Coefficient for relationship between coupled variables.
- VariableValue & _drive_var

  Reference to the coupled non-linear variable that is driving.
- bool _gaining

  Boolean to mark whether the driving force is gaining or losing (True = gaining)
- Real _coef

  Coefficient for the strength or rate of the driving force.
- Real _driving_value

  Value the coupled variable is driving towards.
- VariableValue & _var

  Reference to the coupled non-linear variable.

5.14.1 Detailed Description

CoupledLDF class object inherits from LinearDrivingForce object.

This class object inherits from the LinearDrivingForce object in DGOSPREY. All public and protected members of this class are required function overrides. The kernel has several protected members including: a boolean for gaining or losing mechanisms, a coefficient for the rate or strength of the driving force, a driving value to where the coupled non-linear variable is driving toward, and the coupled non-linear variable.

Additionally, this object couples the driving value to other non-linear variables

Note

To create a specific linear driving force kernel, inherit from this class and use other non-linear variables or material properties to change the protected member values to reflect the physics for your problem.

Definition at line 64 of file CoupledLDF.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 CoupledLDF::CoupledLDF (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.14.3 Member Function Documentation

5.14.3.1 virtual Real CoupledLDF::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from LinearDrivingForce.

5.14.3.2 virtual Real CoupledLDF::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from LinearDrivingForce.

5.14.4 Member Data Documentation

5.14.4.1 Real LinearDrivingForce::_coef [protected], [inherited]

Coefficient for the strength or rate of the driving force.

Definition at line 77 of file LinearDrivingForce.h.

5.14.4.2 Real CoupledLDF::_drive_coef [protected]

Coefficient for relationship between coupled variables.

Definition at line 80 of file CoupledLDF.h.

5.14.4.3 VariableValue& CoupledLDF::_drive_var [protected]

Reference to the coupled non-linear variable that is driving.

Definition at line 81 of file CoupledLDF.h.

5.14.4.4 Real LinearDrivingForce::_driving_value [protected], [inherited]

Value the coupled variable is driving towards.

Definition at line 78 of file LinearDrivingForce.h.

5.14.4.5 bool LinearDrivingForce::_gaining [protected], [inherited]

Boolean to mark whether the driving force is gaining or losing (True = gaining)

Definition at line 76 of file LinearDrivingForce.h.

5.14.4.6 VariableValue& LinearDrivingForce::_var [protected], [inherited]

Reference to the coupled non-linear variable.

Definition at line 79 of file LinearDrivingForce.h.

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

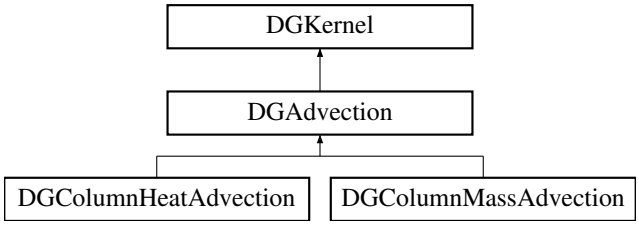
- CoupledLDF.h

5.15 DGAdvection Class Reference

DGAdvection class object inherits from DGKernel object.

`#include <DGAdvection.h>`

Inheritance diagram for DGAdvection:



Public Member Functions

- DGAdvection (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

  Required Jacobian function for DG kernels in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Vector of velocity.
- Real _vx

  x-component of velocity (optional - set in input file)
- Real _vy

  y-component of velocity (optional - set in input file)
- Real _vz

  z-component of velocity (optional - set in input file)

5.15.1 Detailed Description

DGAdvection class object inherits from DGKernel object.

This class object inherits from the DGKernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of advection physics in the MOOSE framework. The only parameter for this kernel is a generic velocity vector, whose components can be set piecewise in the input file or by inheriting from this base class and manually altering the velocity vector.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 66 of file DGAdvection.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 DGAdvection::DGAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.15.3 Member Function Documentation

5.15.3.1 virtual Real DGAdvection::computeQpJacobian (Moose::DGJacobianType *type*) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnHeatAdvection, and DGColumnMassAdvection.

5.15.3.2 virtual Real DGAdvection::computeQpResidual (Moose::DGResidualType *type*) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnHeatAdvection, and DGColumnMassAdvection.

5.15.4 Member Data Documentation

5.15.4.1 RealVectorValue DGAdvection::_velocity [protected]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

5.15.4.2 Real DGAdvection::_vx [protected]

x-component of velocity (optional - set in input file)

Definition at line 83 of file DGAdvection.h.

5.15.4.3 Real DGAdvection::_vy [protected]

y-component of velocity (optional - set in input file)

Definition at line 84 of file DGAdvection.h.

5.15.4.4 Real DGAdvection::_vz [protected]

z-component of velocity (optional - set in input file)

Definition at line 85 of file DGAdvection.h.

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

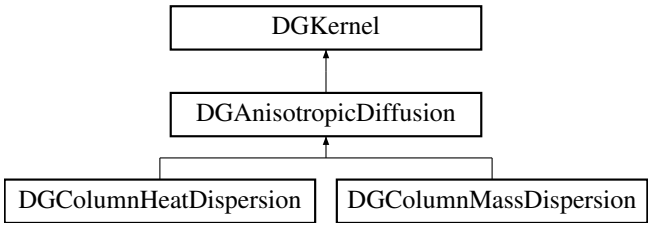
- DGAdvection.h

5.16 DGAnisotropicDiffusion Class Reference

DGAnisotropicDiffusion class object inherits from DGKernel object.

`#include <DGAnisotropicDiffusion.h>`

Inheritance diagram for DGAnisotropicDiffusion:



Public Member Functions

- DGAnisotropicDiffusion (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

  Required Jacobian function for DG kernels in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term for gradient jumps between the solution and test functions.
- Real _sigma

  Penalty term applied to element size.
- RealTensorValue _Diffusion

  Diffusion tensor matrix parameter.
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz

5.16.1 Detailed Description

DGAnisotropicDiffusion class object inherits from DGKernel object.

This class object inherits from the DGKernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of advection physics in the MOOSE framework. The only parameter for this kernel is a diffusion tensor, whose components can be set piecewise in the input file or by inheriting from this base class and manually altering the tensor matrix.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 66 of file DGAnisotropicDiffusion.h.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 DGAnisotropicDiffusion::DGAnisotropicDiffusion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.16.3 Member Function Documentation

5.16.3.1 virtual Real DGAnisotropicDiffusion::computeQpJacobian (Moose::DGJacobianType *type*) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnMassDispersion, and DGColumnHeatDispersion.

5.16.3.2 virtual Real DGAnisotropicDiffusion::computeQpResidual (Moose::DGResidualType *type*) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnMassDispersion, and DGColumnHeatDispersion.

5.16.4 Member Data Documentation

5.16.4.1 RealTensorValue DGAnisotropicDiffusion::_Diffusion [protected]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

5.16.4.2 Real DGAnisotropicDiffusion::_Dxx [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.16.4.3 Real DGAnisotropicDiffusion::_Dxy [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.16.4.4 Real DGAnisotropicDiffusion::_Dxz [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.16.4.5 Real DGAnisotropicDiffusion::_Dyx [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.16.4.6 Real DGAnisotropicDiffusion::_Dyy [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.16.4.7 Real DGAnisotropicDiffusion::_Dyz [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.16.4.8 Real DGAnisotropicDiffusion::_Dzx [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.16.4.9 Real DGAnisotropicDiffusion::_Dzy [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.16.4.10 Real DGAnisotropicDiffusion::_Dzz [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.16.4.11 Real DGAnisotropicDiffusion::_epsilon [protected]

Penalty term for gradient jumps between the solution and test functions.

Definition at line 82 of file DGAnisotropicDiffusion.h.

5.16.4.12 Real DGAnisotropicDiffusion::_sigma [protected]

Penalty term applied to element size.

Definition at line 83 of file DGAnisotropicDiffusion.h.

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

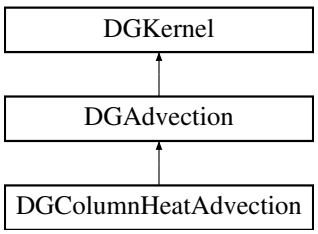
- DGAnisotropicDiffusion.h

5.17 DGColumnHeatAdvection Class Reference

DGColumnHeatAdvection class object inherits from DGAdvection object.

`#include <DGColumnHeatAdvection.h>`

Inheritance diagram for DGColumnHeatAdvection:



Public Member Functions

- DGColumnHeatAdvection (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

  Required Jacobian function for DG kernels in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Vector of velocity.
- Real _vx

x-component of velocity (optional - set in input file)

- Real _vy

  y-component of velocity (optional - set in input file)

- Real _vz

  z-component of velocity (optional - set in input file)

Private Attributes

- const MaterialProperty< Real > & _vel

  Reference to the velocity material property.

- const MaterialProperty< Real > & _gas_density

  Reference to the gas density material property.

- const MaterialProperty< Real > & _gas_heat_capacity

  Reference to the gas heat capacity material property.

5.17.1 Detailed Description

DGColumnHeatAdvection class object inherits from DGAdvection object.

This class object inherits from the DGAdvection object in DGOSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the heat advection physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes velocity vector.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 64 of file DGColumnHeatAdvection.h.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 DGColumnHeatAdvection::DGColumnHeatAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.17.3 Member Function Documentation

5.17.3.1 virtual Real DGColumnHeatAdvection::computeQpJacobian (Moose::DGJacobianType *type*) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAdvection.

5.17.3.2 virtual Real DGColumnHeatAdvection::computeQpResidual (Moose::DGResidualType *type*) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAdvection.

5.17.4 Member Data Documentation

5.17.4.1 const MaterialProperty<Real>& DGColumnHeatAdvection::_gas_density [private]

Reference to the gas density material property.

Definition at line 82 of file DGColumnHeatAdvection.h.

5.17.4.2 const MaterialProperty<Real>& DGColumnHeatAdvection::_gas_heat_capacity [private]

Reference to the gas heat capacity material property.

Definition at line 83 of file DGColumnHeatAdvection.h.

5.17.4.3 const MaterialProperty<Real>& DGColumnHeatAdvection::_vel [private]

Reference to the velocity material property.

Definition at line 81 of file DGColumnHeatAdvection.h.

5.17.4.4 RealVectorValue DGAdvection::_velocity [protected], [inherited]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

5.17.4.5 Real DGAdvection::_vx [protected], [inherited]

x-component of velocity (optional - set in input file)

Definition at line 83 of file DGAdvection.h.

5.17.4.6 Real DGAdvection::_vy [protected], [inherited]

y-component of velocity (optional - set in input file)

Definition at line 84 of file DGAdvection.h.

5.17.4.7 Real DGAdvection::_vz [protected], [inherited]

z-component of velocity (optional - set in input file)

Definition at line 85 of file DGAdvection.h.

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

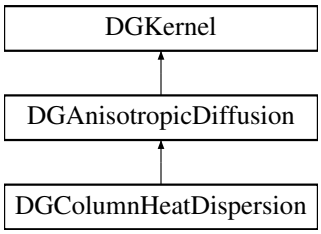
- DGColumnHeatAdvection.h

5.18 DGColumnHeatDispersion Class Reference

DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object.

`#include <DGColumnHeatDispersion.h>`

Inheritance diagram for DGColumnHeatDispersion:



Public Member Functions

- DGColumnHeatDispersion (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

  Required Jacobian function for DG kernels in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term for gradient jumps between the solution and test functions.
- Real _sigma

  Penalty term applied to element size.
- RealTensorValue _Diffusion

  Diffusion tensor matrix parameter.
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz

Private Attributes

- const MaterialProperty< Real > & _conductivity

  Reference to the thermal conductivity material property.

5.18.1 Detailed Description

DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object.

This class object inherits from the DGAnisotropicDiffusion object in OSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the heat dispersion physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes diffusion tensor.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 64 of file DGColumnHeatDispersion.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 DGColumnHeatDispersion::DGColumnHeatDispersion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.18.3 Member Function Documentation

5.18.3.1 virtual Real DGColumnHeatDispersion::computeQpJacobian (Moose::DGJacobianType *type*) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAnisotropicDiffusion.

5.18.3.2 virtual Real DGColumnHeatDispersion::computeQpResidual (Moose::DGResidualType *type*) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAnisotropicDiffusion.

5.18.4 Member Data Documentation

5.18.4.1 const MaterialProperty<Real>& DGColumnHeatDispersion::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 81 of file DGColumnHeatDispersion.h.

5.18.4.2 RealTensorValue DGAnisotropicDiffusion::_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

5.18.4.3 Real DGAnisotropicDiffusion::_Dxx [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.18.4.4 Real DGAnisotropicDiffusion::_Dxy [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.18.4.5 Real DGAnisotropicDiffusion::_Dxz [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.18.4.6 Real DGAnisotropicDiffusion::_Dyx [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.18.4.7 Real DGAnisotropicDiffusion::_Dyy [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.18.4.8 Real DGAnisotropicDiffusion::_Dyz [protected],[inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.18.4.9 Real DGAnisotropicDiffusion::_Dzx [protected],[inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.18.4.10 Real DGAnisotropicDiffusion::_Dzy [protected],[inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.18.4.11 Real DGAnisotropicDiffusion::_Dzz [protected],[inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.18.4.12 Real DGAnisotropicDiffusion::_epsilon [protected],[inherited]

Penalty term for gradient jumps between the solution and test functions.

Definition at line 82 of file DGAnisotropicDiffusion.h.

5.18.4.13 Real DGAnisotropicDiffusion::_sigma [protected],[inherited]

Penalty term applied to element size.

Definition at line 83 of file DGAnisotropicDiffusion.h.

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

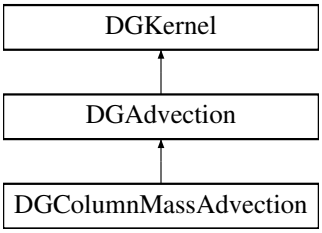
- DGColumnHeatDispersion.h

5.19 DGColumnMassAdvection Class Reference

DGColumnMassAdvection class object inherits from DGAdvection object.

`#include <DGColumnMassAdvection.h>`

Inheritance diagram for DGColumnMassAdvection:



Public Member Functions

- DGColumnMassAdvection (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

  Required Jacobian function for DG kernels in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Vector of velocity.
- Real _vx

  x-component of velocity (optional - set in input file)
- Real _vy

  y-component of velocity (optional - set in input file)
- Real _vz

  z-component of velocity (optional - set in input file)

Private Attributes

- const MaterialProperty< Real > & _vel

  Reference to the velocity material property.

5.19.1 Detailed Description

DGColumnMassAdvection class object inherits from DGAdvection object.

This class object inherits from the DGAdvection object in DGOSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the heat advection physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes velocity vector.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 63 of file DGColumnMassAdvection.h.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 DGColumnMassAdvection::DGColumnMassAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.19.3 Member Function Documentation

5.19.3.1 virtual Real DGColumnMassAdvection::computeQpJacobian (Moose::DGJacobianType *type*) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAdvection.

5.19.3.2 virtual Real DGColumnMassAdvection::computeQpResidual (Moose::DGResidualType *type*) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAdvection.

5.19.4 Member Data Documentation

5.19.4.1 const MaterialProperty<Real>& DGColumnMassAdvection::_vel [private]

Reference to the velocity material property.

Definition at line 80 of file DGColumnMassAdvection.h.

5.19.4.2 RealVectorValue DGAdvection::_velocity [protected], [inherited]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

5.19.4.3 Real DGAdvection::_vx [protected], [inherited]

x-component of velocity (optional - set in input file)

Definition at line 83 of file DGAdvection.h.

5.19.4.4 Real DGAdvection::_vy [protected], [inherited]

y-component of velocity (optional - set in input file)

Definition at line 84 of file DGAdvection.h.

5.19.4.5 Real DGAdvection::_vz [protected], [inherited]

z-component of velocity (optional - set in input file)

Definition at line 85 of file DGAdvection.h.

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

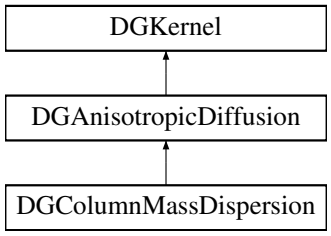
- DGColumnMassAdvection.h

5.20 DGColumnMassDispersion Class Reference

DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object.

`#include <DGColumnMassDispersion.h>`

Inheritance diagram for DGColumnMassDispersion:



Public Member Functions

- DGColumnMassDispersion (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual (Moose::DGResidualType type)

Required residual function for DG kernels in MOOSE.

- virtual Real computeQpJacobian (Moose::DGJacobianType type)

  Required Jacobian function for DG kernels in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term for gradient jumps between the solution and test functions.
- Real _sigma

  Penalty term applied to element size.
- RealTensorValue _Diffusion

  Diffusion tensor matrix parameter.
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz

Private Attributes

- unsigned int _index

  Index of the species of interest for this kernel.
- const MaterialProperty < std::vector< Real > > & _dispersion

  Reference to the axial dispersion material property.
- const MaterialProperty < std::vector< Real > > & _molecular_diffusion

  Reference to the molecular diffusion material property.

5.20.1 Detailed Description

DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object.

This class object inherits from the DGAnisotropicDiffusion object in OSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the mass dispersion physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes diffusion tensor.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 65 of file DGColumnMassDispersion.h.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 DGColumnMassDispersion::DGColumnMassDispersion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.20.3 Member Function Documentation

5.20.3.1 virtual Real DGColumnMassDispersion::computeQpJacobian (Moose::DGJacobianType *type*) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAnisotropicDiffusion.

5.20.3.2 virtual Real DGColumnMassDispersion::computeQpResidual (Moose::DGResidualType *type*) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAnisotropicDiffusion.

5.20.4 Member Data Documentation

5.20.4.1 RealTensorValue DGAnisotropicDiffusion::_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

5.20.4.2 const MaterialProperty<std::vector<Real> >& DGColumnMassDispersion::_dispersion [private]

Reference to the axial dispersion material property.

Definition at line 83 of file DGColumnMassDispersion.h.

5.20.4.3 Real DGAnisotropicDiffusion::_Dxx [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.20.4.4 Real DGAnisotropicDiffusion::_Dxy [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.20.4.5 Real DGAnisotropicDiffusion::_Dxz [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.20.4.6 Real DGAnisotropicDiffusion::_Dyx [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.20.4.7 Real DGAnisotropicDiffusion::_Dyy [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.20.4.8 Real DGAnisotropicDiffusion::_Dyz [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.20.4.9 Real DGAnisotropicDiffusion::_Dzx [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.20.4.10 Real DGAnisotropicDiffusion::_Dzy [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.20.4.11 Real DGAnisotropicDiffusion::_Dzz [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.20.4.12 Real DGAnisotropicDiffusion::_epsilon [protected], [inherited]

Penalty term for gradient jumps between the solution and test functions.

Definition at line 82 of file DGAnisotropicDiffusion.h.

5.20.4.13 unsigned int DGColumnMassDispersion::_index [private]

Index of the species of interest for this kernel.

Definition at line 82 of file DGColumnMassDispersion.h.

5.20.4.14 const MaterialProperty<std::vector<Real> >& DGColumnMassDispersion::_molecular_diffusion [private]

Reference to the molecular diffusion material property.

Definition at line 84 of file DGColumnMassDispersion.h.

5.20.4.15 Real DGAnisotropicDiffusion::_sigma [protected], [inherited]

Penalty term applied to element size.

Definition at line 83 of file DGAnisotropicDiffusion.h.

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

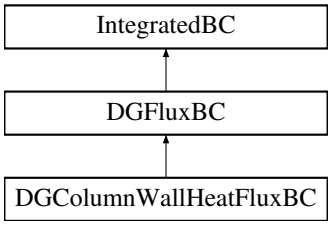
- DGColumnMassDispersion.h

5.21 DGColumnWallHeatFluxBC Class Reference

DGColumnWallHeatFluxBC class object inherits from DGFluxBC object.

`#include <DGColumnWallHeatFluxBC.h>`

Inheritance diagram for DGColumnWallHeatFluxBC:



Public Member Functions

- DGColumnWallHeatFluxBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

Private Attributes

- VariableValue & _wall_temp

  Reference to the coupled variable for wall temperature of the column.
- const MaterialProperty< Real > & _bed_wall_transfer_coeff

  Reference to the bed-wall transfer coefficient material property.
- const MaterialProperty< Real > & _conductivity

  Reference to the thermal conductivity material property.

5.21.1 Detailed Description

DGColumnWallHeatFluxBC class object inherits from DGFluxBC object.

This class object inherits from the DGFluxBC object (see DGFluxBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxBC functions.

Definition at line 62 of file DGColumnWallHeatFluxBC.h.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 DGColumnWallHeatFluxBC::DGColumnWallHeatFluxBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.21.3 Member Function Documentation

5.21.3.1 virtual Real DGColumnWallHeatFluxBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxBC.

5.21.3.2 virtual Real DGColumnWallHeatFluxBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxBC.

5.21.4 Member Data Documentation

5.21.4.1 const MaterialProperty<Real>& DGColumnWallHeatFluxBC::_bed_wall_transfer_coeff [private]

Reference to the bed-wall transfer coefficient material property.

Definition at line 81 of file DGColumnWallHeatFluxBC.h.

5.21.4.2 const MaterialProperty<Real>& DGColumnWallHeatFluxBC::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 82 of file DGColumnWallHeatFluxBC.h.

5.21.4.3 RealTensorValue DGFluxBC::_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

5.21.4.4 Real DGFluxBC::_Dxx [protected], [inherited]

Definition at line 89 of file DGFluxBC.h.

5.21.4.5 Real DGFluxBC::_Dxy [protected], [inherited]

Definition at line 89 of file DGFluxBC.h.

5.21.4.6 Real DGFluxBC::_Dxz [protected], [inherited]

Definition at line 89 of file DGFluxBC.h.

5.21.4.7 Real DGFluxBC::_Dyx [protected], [inherited]

Definition at line 90 of file DGFluxBC.h.

5.21.4.8 Real DGFluxBC::_Dyy [protected], [inherited]

Definition at line 90 of file DGFluxBC.h.

5.21.4.9 Real DGFluxBC::_Dyz [protected], [inherited]

Definition at line 90 of file DGFluxBC.h.

5.21.4.10 Real DGFluxBC::_Dzx [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.21.4.11 Real DGFluxBC::_Dzy [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.21.4.12 Real DGFluxBC::_Dzz [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.21.4.13 Real DGFluxBC::_u_input [protected],[inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

5.21.4.14 RealVectorValue DGFluxBC::_velocity [protected],[inherited]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

5.21.4.15 Real DGFluxBC::_vx [protected],[inherited]

Definition at line 85 of file DGFluxBC.h.

5.21.4.16 Real DGFluxBC::_vy [protected],[inherited]

Definition at line 86 of file DGFluxBC.h.

5.21.4.17 Real DGFluxBC::_vz [protected],[inherited]

Definition at line 87 of file DGFluxBC.h.

5.21.4.18 VariableValue& DGColumnWallHeatFluxBC::_wall_temp [private]

Reference to the coupled variable for wall temperature of the column.

Definition at line 80 of file DGColumnWallHeatFluxBC.h.

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

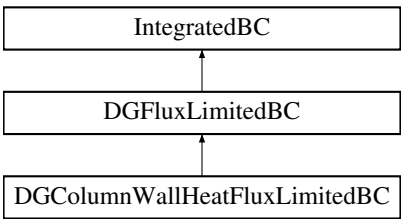
- DGColumnWallHeatFluxBC.h

5.22 DGColumnWallHeatFluxLimitedBC Class Reference

DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

`#include <DGColumnWallHeatFluxLimitedBC.h>`

Inheritance diagram for DGColumnWallHeatFluxLimitedBC:



Public Member Functions

- DGColumnWallHeatFluxLimitedBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required function override for BC objects in MOOSE.
- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.
- Real _sigma

  Penalty term based on the size of the element at the boundary.
- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

Private Attributes

- VariableValue & _wall_temp

  Reference to the coupled variable for wall temperature of the column.
- const MaterialProperty< Real > & _bed_wall_transfer_coeff

  Reference to the bed-wall transfer coefficient material property.
- const MaterialProperty< Real > & _conductivity

  Reference to the thermal conductivity material property.

5.22.1 Detailed Description

DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

This class object inherits from the DGFluxLimitedBC object (see DGFluxLimitedBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxLimitedBC functions.

Definition at line 55 of file DGColumnWallHeatFluxLimitedBC.h.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 DGColumnWallHeatFluxLimitedBC::DGColumnWallHeatFluxLimitedBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.22.3 Member Function Documentation

5.22.3.1 virtual Real DGColumnWallHeatFluxLimitedBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxLimitedBC.

5.22.3.2 virtual Real DGColumnWallHeatFluxLimitedBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxLimitedBC.

5.22.4 Member Data Documentation

5.22.4.1 const MaterialProperty<Real>& DGColumnWallHeatFluxLimitedBC::_bed_wall_transfer_coeff [private]

Reference to the bed-wall transfer coefficient material property.

Definition at line 74 of file DGColumnWallHeatFluxLimitedBC.h.

5.22.4.2 const MaterialProperty<Real>& DGColumnWallHeatFluxLimitedBC::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 75 of file DGColumnWallHeatFluxLimitedBC.h.

5.22.4.3 RealTensorValue DGFluxLimitedBC::_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

5.22.4.4 Real DGFluxLimitedBC::_Dxx [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.22.4.5 Real DGFluxLimitedBC::_Dxy [protected],[inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.22.4.6 Real DGFluxLimitedBC::_Dxz [protected],[inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.22.4.7 Real DGFluxLimitedBC::_Dyx [protected],[inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.22.4.8 Real DGFluxLimitedBC::_Dyy [protected],[inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.22.4.9 Real DGFluxLimitedBC::_Dyz [protected],[inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.22.4.10 Real DGFluxLimitedBC::_Dzx [protected],[inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.22.4.11 Real DGFluxLimitedBC::_Dzy [protected],[inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.22.4.12 Real DGFluxLimitedBC::_Dzz [protected],[inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.22.4.13 Real DGFluxLimitedBC::_epsilon [protected],[inherited]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

5.22.4.14 Real DGFluxLimitedBC::_sigma [protected],[inherited]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

5.22.4.15 Real DGFluxLimitedBC::_u_input [protected],[inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

5.22.4.16 RealVectorValue DGFluxLimitedBC::_velocity [protected],[inherited]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

5.22.4.17 Real DGFluxLimitedBC::_vx [protected],[inherited]

Definition at line 81 of file DGFluxLimitedBC.h.

5.22.4.18 Real DGFluxLimitedBC::_vy [protected],[inherited]

Definition at line 82 of file DGFluxLimitedBC.h.

5.22.4.19 Real DGFluxLimitedBC::_vz [protected],[inherited]

Definition at line 83 of file DGFluxLimitedBC.h.

5.22.4.20 VariableValue& DGColumnWallHeatFluxLimitedBC::_wall_temp [private]

Reference to the coupled variable for wall temperature of the column.

Definition at line 73 of file DGColumnWallHeatFluxLimitedBC.h.

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

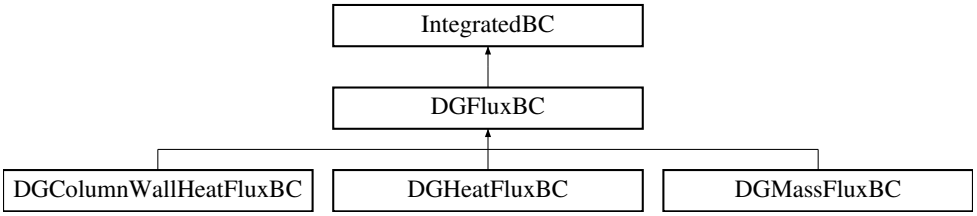
- DGColumnWallHeatFluxLimitedBC.h

5.23 DGFluxBC Class Reference

DGFluxBC class object inherits from IntegratedBC object.

`#include <DGFluxBC.h>`

Inheritance diagram for DGFluxBC:



Public Member Functions

- DGFluxBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required function override for BC objects in MOOSE.
- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx

- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

5.23.1 Detailed Description

DGFluxBC class object inherits from IntegratedBC object.

This class object inherits from the IntegratedBC object. All public and protected members of this class are required function overrides. The flux BC uses the velocity and diffusivity in the system to apply a boundary condition based on whether or not material is leaving or entering the boundary.

Definition at line 63 of file DGFluxBC.h.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 DGFluxBC::DGFluxBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.23.3 Member Function Documentation

5.23.3.1 virtual Real DGFluxBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnWallHeatFluxBC, DGHeatFluxBC, and DGMassFluxBC.

5.23.3.2 virtual Real DGFluxBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnWallHeatFluxBC, DGHeatFluxBC, and DGMassFluxBC.

5.23.4 Member Data Documentation

5.23.4.1 RealTensorValue DGFluxBC::_Diffusion [protected]

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

5.23.4.2 Real DGFluxBC::_Dxx [protected]

Definition at line 89 of file DGFluxBC.h.

5.23.4.3 Real DGFluxBC::_Dxy [protected]

Definition at line 89 of file DGFluxBC.h.

5.23.4.4 Real DGFluxBC::_Dxz [protected]

Definition at line 89 of file DGFluxBC.h.

5.23.4.5 Real DGFluxBC::_Dyx [protected]

Definition at line 90 of file DGFluxBC.h.

5.23.4.6 Real DGFluxBC::_Dyy [protected]

Definition at line 90 of file DGFluxBC.h.

5.23.4.7 Real DGFluxBC::_Dyz [protected]

Definition at line 90 of file DGFluxBC.h.

5.23.4.8 Real DGFluxBC::_Dzx [protected]

Definition at line 91 of file DGFluxBC.h.

5.23.4.9 Real DGFluxBC::_Dzy [protected]

Definition at line 91 of file DGFluxBC.h.

5.23.4.10 Real DGFluxBC::_Dzz [protected]

Definition at line 91 of file DGFluxBC.h.

5.23.4.11 Real DGFluxBC::_u_input [protected]

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

5.23.4.12 RealVectorValue DGFluxBC::_velocity [protected]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

5.23.4.13 Real DGFluxBC::_vx [protected]

Definition at line 85 of file DGFluxBC.h.

5.23.4.14 Real DGFluxBC::_vy [protected]

Definition at line 86 of file DGFluxBC.h.

5.23.4.15 Real DGFluxBC::_vz [protected]

Definition at line 87 of file DGFluxBC.h.

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

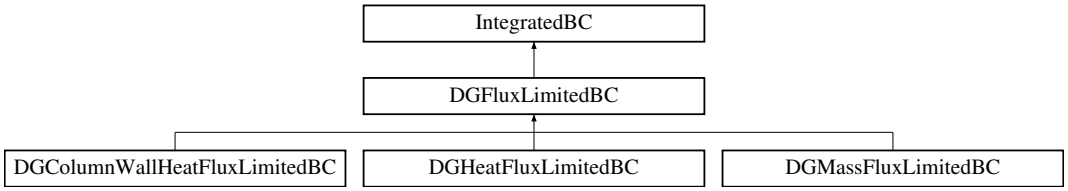
- DGFluxBC.h

5.24 DGFluxLimitedBC Class Reference

DGFluxLimitedBC class object inherits from IntegratedBC object.

`#include <DGFluxLimitedBC.h>`

Inheritance diagram for DGFluxLimitedBC:



Public Member Functions

- DGFluxLimitedBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required function override for BC objects in MOOSE.
- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.
- Real _sigma

  Penalty term based on the size of the element at the boundary.
- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

5.24.1 Detailed Description

DGFluxLimitedBC class object inherits from IntegratedBC object.

This class object inherits from the IntegratedBC object. All public and protected members of this class are required function overrides.

Definition at line 55 of file DGFluxLimitedBC.h.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 DGFluxLimitedBC::DGFluxLimitedBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.24.3 Member Function Documentation

5.24.3.1 virtual Real DGFluxLimitedBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnWallHeatFluxLimitedBC, DGHeatFluxLimitedBC, and DGMassFluxLimitedBC.

5.24.3.2 virtual Real DGFluxLimitedBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnWallHeatFluxLimitedBC, DGHeatFluxLimitedBC, and DGMassFluxLimitedBC.

5.24.4 Member Data Documentation

5.24.4.1 RealTensorValue DGFluxLimitedBC::_Diffusion [protected]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

5.24.4.2 Real DGFluxLimitedBC::_Dxx [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

5.24.4.3 Real DGFluxLimitedBC::_Dxy [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

5.24.4.4 Real DGFluxLimitedBC::_Dxz [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

5.24.4.5 Real DGFluxLimitedBC::_Dyx [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

5.24.4.6 Real DGFluxLimitedBC::_Dyy [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

5.24.4.7 Real DGFluxLimitedBC::_Dyz [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

5.24.4.8 Real DGFluxLimitedBC::_Dzx [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

5.24.4.9 Real DGFluxLimitedBC::_Dzy [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

5.24.4.10 Real DGFluxLimitedBC::_Dzz [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

5.24.4.11 Real DGFluxLimitedBC::_epsilon [protected]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

5.24.4.12 Real DGFluxLimitedBC::_sigma [protected]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

5.24.4.13 Real DGFluxLimitedBC::_u_input [protected]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

5.24.4.14 RealVectorValue DGFluxLimitedBC::_velocity [protected]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

5.24.4.15 Real DGFluxLimitedBC::_vx [protected]

Definition at line 81 of file DGFluxLimitedBC.h.

5.24.4.16 Real DGFluxLimitedBC::_vy [protected]

Definition at line 82 of file DGFluxLimitedBC.h.

5.24.4.17 Real DGFluxLimitedBC::_vz [protected]

Definition at line 83 of file DGFluxLimitedBC.h.

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

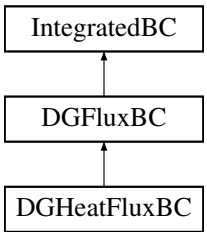
- DGFluxLimitedBC.h

5.25 DGHeatFluxBC Class Reference

DGHeatFluxBC class object inherits from DGFluxBC object.

`#include <DGHeatFluxBC.h>`

Inheritance diagram for DGHeatFluxBC:



Public Member Functions

- DGHeatFluxBC (const InputParameters &parameters)
  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()
  Required function override for BC objects in MOOSE.
- virtual Real computeQpJacobian ()
  Required function override for BC objects in MOOSE.

Protected Attributes

- RealVectorValue _velocity
  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion
  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input
  Value of the non-linear variable at the input of the boundary.

Private Attributes

- Real _input_temperature
  Value of the column temperature at the inlet of the system.
- const MaterialProperty< Real > & _vel
  Reference to the velocity material property.
- const MaterialProperty< Real > & _gas_density
  Reference to the gas density material property.
- const MaterialProperty< Real > & _gas_heat_capacity
  Reference to the gas heat capacity material property.
- const MaterialProperty< Real > & _conductivity
  Reference to the thermal conductivity material property.

5.25.1 Detailed Description

DGHeatFluxBC class object inherits from DGFluxBC object.

This class object inherits from the DGFluxBC object (see DGFluxBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxBC functions.

Definition at line 62 of file DGHeatFluxBC.h.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 DGHeatFluxBC::DGHeatFluxBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.25.3 Member Function Documentation

5.25.3.1 virtual Real DGHeatFluxBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxBC.

5.25.3.2 virtual Real DGHeatFluxBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxBC.

5.25.4 Member Data Documentation

5.25.4.1 const MaterialProperty<Real>& DGHeatFluxBC::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 85 of file DGHeatFluxBC.h.

5.25.4.2 RealTensorValue DGFluxBC::_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

5.25.4.3 Real DGFluxBC::_Dxx [protected], [inherited]

Definition at line 89 of file DGFluxBC.h.

5.25.4.4 Real DGFluxBC::_Dxy [protected], [inherited]

Definition at line 89 of file DGFluxBC.h.

5.25.4.5 Real DGFluxBC::_Dxz [protected], [inherited]

Definition at line 89 of file DGFluxBC.h.

5.25.4.6 Real DGFluxBC::_Dyx [protected],[inherited]

Definition at line 90 of file DGFluxBC.h.

5.25.4.7 Real DGFluxBC::_Dyy [protected],[inherited]

Definition at line 90 of file DGFluxBC.h.

5.25.4.8 Real DGFluxBC::_Dyz [protected],[inherited]

Definition at line 90 of file DGFluxBC.h.

5.25.4.9 Real DGFluxBC::_Dzx [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.25.4.10 Real DGFluxBC::_Dzy [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.25.4.11 Real DGFluxBC::_Dzz [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.25.4.12 const MaterialProperty<Real>& DGHeatFluxBC::_gas_density [private]

Reference to the gas density material property.

Definition at line 83 of file DGHeatFluxBC.h.

5.25.4.13 const MaterialProperty<Real>& DGHeatFluxBC::_gas_heat_capacity [private]

Reference to the gas heat capacity material property.

Definition at line 84 of file DGHeatFluxBC.h.

5.25.4.14 Real DGHeatFluxBC::_input_temperature [private]

Value of the column temperature at the inlet of the system.

Definition at line 80 of file DGHeatFluxBC.h.

5.25.4.15 Real DGFluxBC::_u_input [protected],[inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

5.25.4.16 const MaterialProperty<Real>& DGHeatFluxBC::_vel [private]

Reference to the velocity material property.

Definition at line 82 of file DGHeatFluxBC.h.

5.25.4.17 RealVectorValue DGFluxBC::_velocity [protected],[inherited]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

5.25.4.18 Real DGFluxBC::_vx [protected],[inherited]

Definition at line 85 of file DGFluxBC.h.

5.25.4.19 Real DGFluxBC::_vy [protected],[inherited]

Definition at line 86 of file DGFluxBC.h.

5.25.4.20 Real DGFluxBC::_vz [protected],[inherited]

Definition at line 87 of file DGFluxBC.h.

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

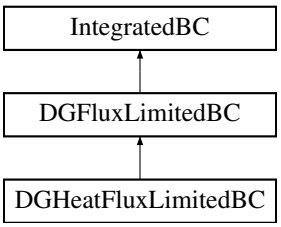
- DGHeatFluxBC.h

5.26 DGHeatFluxLimitedBC Class Reference

DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

`#include <DGHeatFluxLimitedBC.h>`

Inheritance diagram for DGHeatFluxLimitedBC:



Public Member Functions

- DGHeatFluxLimitedBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required function override for BC objects in MOOSE.
- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.
- Real _sigma

  Penalty term based on the size of the element at the boundary.
- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx

- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

Private Attributes

- Real _input_temperature

  Value of the column temperature at the inlet of the system.
- const MaterialProperty< Real > & _vel

  Reference to the velocity material property.
- const MaterialProperty< Real > & _gas_density

  Reference to the gas density material property.
- const MaterialProperty< Real > & _gas_heat_capacity

  Reference to the gas heat capacity material property.
- const MaterialProperty< Real > & _conductivity

  Reference to the thermal conductivity material property.

5.26.1 Detailed Description

DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

This class object inherits from the DGFluxLimitedBC object (see DGFluxLimitedBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxLimitedBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxLimitedBC functions.

Definition at line 55 of file DGHeatFluxLimitedBC.h.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 DGHeatFluxLimitedBC::DGHeatFluxLimitedBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.26.3 Member Function Documentation

5.26.3.1 virtual Real DGHeatFluxLimitedBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxLimitedBC.

5.26.3.2 virtual Real DGHeatFluxLimitedBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxLimitedBC.

5.26.4 Member Data Documentation

5.26.4.1 const MaterialProperty<Real>& DGHeatFluxLimitedBC::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 78 of file DGHeatFluxLimitedBC.h.

5.26.4.2 RealTensorValue DGFluxLimitedBC::_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

5.26.4.3 Real DGFluxLimitedBC::_Dxx [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.26.4.4 Real DGFluxLimitedBC::_Dxy [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.26.4.5 Real DGFluxLimitedBC::_Dxz [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.26.4.6 Real DGFluxLimitedBC::_Dyx [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.26.4.7 Real DGFluxLimitedBC::_Dyy [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.26.4.8 Real DGFluxLimitedBC::_Dyz [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.26.4.9 Real DGFluxLimitedBC::_Dzx [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.26.4.10 Real DGFluxLimitedBC::_Dzy [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.26.4.11 Real DGFluxLimitedBC::_Dzz [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.26.4.12 Real DGFluxLimitedBC::_epsilon [protected], [inherited]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

5.26.4.13 const MaterialProperty<Real>& DGHeatFluxLimitedBC::_gas_density [private]

Reference to the gas density material property.

Definition at line 76 of file DGHeatFluxLimitedBC.h.

5.26.4.14 const MaterialProperty<Real>& DGHeatFluxLimitedBC::_gas_heat_capacity [private]

Reference to the gas heat capacity material property.

Definition at line 77 of file DGHeatFluxLimitedBC.h.

5.26.4.15 Real DGHeatFluxLimitedBC::_input_temperature [private]

Value of the column temperature at the inlet of the system.

Definition at line 73 of file DGHeatFluxLimitedBC.h.

5.26.4.16 Real DGFluxLimitedBC::_sigma [protected], [inherited]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

5.26.4.17 Real DGFluxLimitedBC::_u_input [protected], [inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

5.26.4.18 const MaterialProperty<Real>& DGHeatFluxLimitedBC::_vel [private]

Reference to the velocity material property.

Definition at line 75 of file DGHeatFluxLimitedBC.h.

5.26.4.19 RealVectorValue DGFluxLimitedBC::_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

5.26.4.20 Real DGFluxLimitedBC::_vx [protected], [inherited]

Definition at line 81 of file DGFluxLimitedBC.h.

5.26.4.21 Real DGFluxLimitedBC::_vy [protected], [inherited]

Definition at line 82 of file DGFluxLimitedBC.h.

5.26.4.22 Real DGFluxLimitedBC::_vz [protected], [inherited]

Definition at line 83 of file DGFluxLimitedBC.h.

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

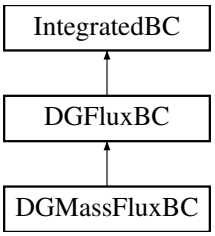
- DGHeatFluxLimitedBC.h

5.27 DGMassFluxBC Class Reference

DGMassFluxBC class object inherits from DGFluxBC object.

`#include <DGMassFluxBC.h>`

Inheritance diagram for DGMassFluxBC:



Public Member Functions

- DGMassFluxBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required function override for BC objects in MOOSE.
- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

Private Attributes

- Real _input_temperature

  Value of the column temperature at the inlet of the system.
- Real _input_pressure

  Value of the column total pressure at the inlet of the system.
- Real _input_molefraction

  Value of the molefraction of the specific species at the inlet of the system.
- const MaterialProperty< Real > & _vel

  Reference to the velocity material property.

- unsigned int _index

  Index of the species of interest at the boundary.
- const MaterialProperty < std::vector< Real > > & _dispersion

  Reference to the dispersion coefficient material property.
- const MaterialProperty < std::vector< Real > > & _molecular_diffusion

  Reference to the molecular diffusion material property.

5.27.1 Detailed Description

DGMassFluxBC class object inherits from DGFluxBC object.

This class object inherits from the DGFluxBC object (see DGFluxBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxBC functions.

Definition at line 62 of file DGMassFluxBC.h.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 DGMassFluxBC::DGMassFluxBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.27.3 Member Function Documentation

5.27.3.1 virtual Real DGMassFluxBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxBC.

5.27.3.2 virtual Real DGMassFluxBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxBC.

5.27.4 Member Data Documentation

5.27.4.1 RealTensorValue DGFluxBC::_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

5.27.4.2 const MaterialProperty<std::vector<Real> >& DGMassFluxBC::_dispersion [private]

Reference to the dispersion coefficient material property.

Definition at line 88 of file DGMassFluxBC.h.

5.27.4.3 Real DGFluxBC::_Dxx [protected],[inherited]

Definition at line 89 of file DGFluxBC.h.

5.27.4.4 Real DGFluxBC::_Dxy [protected],[inherited]

Definition at line 89 of file DGFluxBC.h.

5.27.4.5 Real DGFluxBC::_Dxz [protected],[inherited]

Definition at line 89 of file DGFluxBC.h.

5.27.4.6 Real DGFluxBC::_Dyx [protected],[inherited]

Definition at line 90 of file DGFluxBC.h.

5.27.4.7 Real DGFluxBC::_Dyy [protected],[inherited]

Definition at line 90 of file DGFluxBC.h.

5.27.4.8 Real DGFluxBC::_Dyz [protected],[inherited]

Definition at line 90 of file DGFluxBC.h.

5.27.4.9 Real DGFluxBC::_Dzx [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.27.4.10 Real DGFluxBC::_Dzy [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.27.4.11 Real DGFluxBC::_Dzz [protected],[inherited]

Definition at line 91 of file DGFluxBC.h.

5.27.4.12 unsigned int DGMassFluxBC::_index [private]

Index of the species of interest at the boundary.

Definition at line 87 of file DGMassFluxBC.h.

5.27.4.13 Real DGMassFluxBC::_input_molefraction [private]

Value of the molefraction of the specific species at the inlet of the system.

Definition at line 84 of file DGMassFluxBC.h.

5.27.4.14 Real DGMassFluxBC::_input_pressure [private]

Value of the column total pressure at the inlet of the system.

Definition at line 82 of file DGMassFluxBC.h.

5.27.4.15 Real DGMassFluxBC::_input_temperature [private]

Value of the column temperature at the inlet of the system.

Definition at line 80 of file DGMassFluxBC.h.

5.27.4.16 const MaterialProperty<std::vector<Real> >& DGMassFluxBC::_molecular_diffusion [private]

Reference to the molecular diffusion material property.

Definition at line 89 of file DGMassFluxBC.h.

5.27.4.17 Real DGFluxBC::_u_input [protected], [inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

5.27.4.18 const MaterialProperty<Real>& DGMassFluxBC::_vel [private]

Reference to the velocity material property.

Definition at line 86 of file DGMassFluxBC.h.

5.27.4.19 RealVectorValue DGFluxBC::_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

5.27.4.20 Real DGFluxBC::_vx [protected], [inherited]

Definition at line 85 of file DGFluxBC.h.

5.27.4.21 Real DGFluxBC::_vy [protected], [inherited]

Definition at line 86 of file DGFluxBC.h.

5.27.4.22 Real DGFluxBC::_vz [protected], [inherited]

Definition at line 87 of file DGFluxBC.h.

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

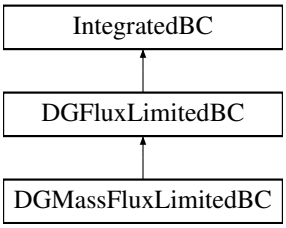
- DGMassFluxBC.h

5.28 DGMassFluxLimitedBC Class Reference

DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object.

`#include <DGMassFluxLimitedBC.h>`

Inheritance diagram for DGMassFluxLimitedBC:



Public Member Functions

- DGMassFluxLimitedBC (const InputParameters &parameters)

  Required constructor for BC objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

- virtual Real computeQpJacobian ()

  Required function override for BC objects in MOOSE.

Protected Attributes

- Real _epsilon

  Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.
- Real _sigma

  Penalty term based on the size of the element at the boundary.
- RealVectorValue _velocity

  Velocity vector in the system or at the boundary.
- RealTensorValue _Diffusion

  Diffusivity tensory in the system or at the boundary.
- Real _vx
- Real _vy
- Real _vz
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz
- Real _u_input

  Value of the non-linear variable at the input of the boundary.

Private Attributes

- Real _input_temperature

  Value of the column temperature at the inlet of the system.
- Real _input_pressure

  Value of the column total pressure at the inlet of the system.
- Real _input_molefraction

  Value of the molefraction of the specific species at the inlet of the system.
- const MaterialProperty< Real > & _vel

  Reference to the velocity material property.
- unsigned int _index

  Index of the species of interest at the boundary.
- const MaterialProperty
  < std::vector< Real > > & _dispersion

  Reference to the dispersion coefficient material property.
- const MaterialProperty
  < std::vector< Real > > & _molecular_diffusion

  Reference to the molecular diffusion material property.

5.28.1 Detailed Description

DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object.

This class object inherits from the DGFluxLimitedBC object (see DGFluxLimitedBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxLimitedBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxLimitedBC functions.

Definition at line 55 of file DGMassFluxLimitedBC.h.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 DGMassFluxLimitedBC::DGMassFluxLimitedBC (const InputParameters & *parameters*)

Required constructor for BC objects in MOOSE.

5.28.3 Member Function Documentation

5.28.3.1 virtual Real DGMassFluxLimitedBC::computeQpJacobian ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxLimitedBC.

5.28.3.2 virtual Real DGMassFluxLimitedBC::computeQpResidual ( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxLimitedBC.

5.28.4 Member Data Documentation

5.28.4.1 RealTensorValue DGFluxLimitedBC::_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

5.28.4.2 const MaterialProperty<std::vector<Real> >& DGMassFluxLimitedBC::_dispersion [private]

Reference to the dispersion coefficient material property.

Definition at line 81 of file DGMassFluxLimitedBC.h.

5.28.4.3 Real DGFluxLimitedBC::_Dxx [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.28.4.4 Real DGFluxLimitedBC::_Dxy [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.28.4.5 Real DGFluxLimitedBC::_Dxz [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

5.28.4.6 Real DGFluxLimitedBC::_Dyx [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.28.4.7 Real DGFluxLimitedBC::_Dyy [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.28.4.8 Real DGFluxLimitedBC::_Dyz [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

5.28.4.9 Real DGFluxLimitedBC::_Dzx [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.28.4.10 Real DGFluxLimitedBC::_Dzy [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.28.4.11 Real DGFluxLimitedBC::_Dzz [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

5.28.4.12 Real DGFluxLimitedBC::_epsilon [protected], [inherited]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

5.28.4.13 unsigned int DGMassFluxLimitedBC::_index [private]

Index of the species of interest at the boundary.

Definition at line 80 of file DGMassFluxLimitedBC.h.

5.28.4.14 Real DGMassFluxLimitedBC::_input_molefraction [private]

Value of the molefraction of the specific species at the inlet of the system.

Definition at line 77 of file DGMassFluxLimitedBC.h.

5.28.4.15 Real DGMassFluxLimitedBC::_input_pressure [private]

Value of the column total pressure at the inlet of the system.

Definition at line 75 of file DGMassFluxLimitedBC.h.

5.28.4.16 Real DGMassFluxLimitedBC::_input_temperature [private]

Value of the column temperature at the inlet of the system.

Definition at line 73 of file DGMassFluxLimitedBC.h.

5.28.4.17 const MaterialProperty<std::vector<Real> >& DGMassFluxLimitedBC::_molecular_diffusion [private]

Reference to the molecular diffusion material property.

Definition at line 82 of file DGMassFluxLimitedBC.h.

5.28.4.18 Real DGFluxLimitedBC::_sigma [protected], [inherited]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

5.28.4.19 Real DGFluxLimitedBC::_u_input [protected],[inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

5.28.4.20 const MaterialProperty<Real>& DGMassFluxLimitedBC::_vel [private]

Reference to the velocity material property.

Definition at line 79 of file DGMassFluxLimitedBC.h.

5.28.4.21 RealVectorValue DGFluxLimitedBC::_velocity [protected],[inherited]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

5.28.4.22 Real DGFluxLimitedBC::_vx [protected],[inherited]

Definition at line 81 of file DGFluxLimitedBC.h.

5.28.4.23 Real DGFluxLimitedBC::_vy [protected],[inherited]

Definition at line 82 of file DGFluxLimitedBC.h.

5.28.4.24 Real DGFluxLimitedBC::_vz [protected],[inherited]

Definition at line 83 of file DGFluxLimitedBC.h.

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

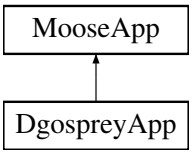
- DGMassFluxLimitedBC.h

5.29 DgospreyApp Class Reference

DgospreyApp inherits from the MooseApp object.

`#include <DgospreyApp.h>`

Inheritance diagram for DgospreyApp:



Public Member Functions

- DgospreyApp (InputParameters parameters)

  DgospreyApp constructor (required)
- virtual ~DgospreyApp ()

  DgospreyApp destructor (requried)

Static Public Member Functions

- static void registerApps ()

  Function to the DgospreyApp into MOOSE (required)

- static void registerObjects (Factory &factory)

  Function to register kernels/objects created in DGOSPREY into the application (required)
- static void associateSyntax (Syntax &syntax, ActionFactory &action_factory)

  Function to associate syntax with the DgospreyApp (required?)

5.29.1 Detailed Description

DgospreyApp inherits from the MooseApp object.

This object defines the required constructors, destructors, and functions that must be a part of every MooseApp based object. All MooseApp objects must be created in this way and override these functions.

Definition at line 82 of file DgospreyApp.h.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 DgospreyApp::DgospreyApp (InputParameters *parameters*)

DgospreyApp constructor (required)

5.29.2.2 virtual DgospreyApp::~DgospreyApp ( ) [virtual]

DgospreyApp destructor (requried)

5.29.3 Member Function Documentation

5.29.3.1 static void DgospreyApp::associateSyntax (Syntax & *syntax,* ActionFactory & *action_factory*) [static]

Function to associate syntax with the DgospreyApp (required?)

I don't know what this is or does or what is actually being registered.

5.29.3.2 static void DgospreyApp::registerApps ( ) [static]

Function to the DgospreyApp into MOOSE (required)

5.29.3.3 static void DgospreyApp::registerObjects (Factory & *factory*) [static]

Function to register kernels/objects created in DGOSPREY into the application (required)

This is the function where the user must register all the kernels and other modules that are to be used in DGOSPREY. Each time a new kernel or other object is created in DGOSPREY, it must be registered here prior to building and running the application. Otherwise, the new functionallity added will not show up or be utilized.

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

- DgospreyApp.h

5.30 FINCH_DATA Struct Reference

Data structure for the FINCH object.

`#include <finch.h>`

Public Attributes

- int d = 0

  Dimension of the problem: 0 = cartesian, 1 = cylindrical, 2 = spherical.

- double dt = 0.0125

  Time step.
- double dt_old = 0.0125

  Previous time step.
- double T = 1.0

  Total time.
- double dz = 0.1

  Space step.
- double L = 1.0

  Total space.
- double s = 1.0

  Char quantity (spherical = 1, cylindrical = length, cartesian = area)
- double t = 0.0

  Current Time.
- double t_old = 0.0

  Previous Time.
- double uT = 0.0

  Total amount of conserved quantity in domain.
- double uT_old = 0.0

  Old Total amount of conserved quantity.
- double uAvg = 0.0

  Average amount of conserved quantity in domain.
- double uAvg_old = 0.0

  Old Average amount of conserved quantity.
- double uIC = 0.0

  Initial condition of Conserved Quantity (if constant)
- double vIC = 1.0

  Initial condition of Velocity (if constant)
- double DIC = 1.0

  Initial condition of Dispersion (if constant)
- double kIC = 1.0

  Initial condition of Reaction (if constant)
- double RIC = 1.0

  Initial condition of the Time Coefficient (if constant)
- double uo = 1.0

  Boundary Value of Conserved Quantity.
- double vo = 1.0

  Boundary Value of Velocity.
- double Do = 1.0

  Boundary Value of Dispersion.
- double ko = 1.0

  Boundary Value of Reaction.
- double Ro = 1.0

  Boundary Value of Time Coefficient.
- double kfn = 1.0

  Film mass transfer coefficient Old.
- double kfnp1 = 1.0

  Film mass transfer coefficient New.
- double lambda_I

  Boundary Coefficient for Implicit Neumann (Calculated at Runtime)
- double lambda_E

Boundary Coefficient for Explicit Neumann (Calculated at Runtime)

- int LN = 10

  Number of nodes.
- bool CN = true

  True if Crank-Nicholson, false if Implicit, never use explicit.
- bool Update = false

  Flag to check if the system needs updating.
- bool Dirichlet = false

  Flag to indicate use of Dirichlet or Neumann starting boundary.
- bool CheckMass = false

  Flag to indicate whether or not mass is to be checked.
- bool ExplicitFlux = false

  Flag to indicate whether or not to use fully explicit flux limiters.
- bool Iterative = true

  Flag to indicate whether to solve directly, or iteratively.
- bool SteadyState = false

  Flag to determine whether or not to solve the steady-state problem.
- bool NormTrack = true

  Flag to determine whether or not to track the norms during simulation.
- double beta = 0.5

  Scheme type indicator: 0.5=CN & 1.0=Implicit; all else NULL.
- double tol_rel = 1e-6

  Relative Tolerance for Convergence.
- double tol_abs = 1e-6

  Absolute Tolerance for Convergence.
- int max_iter = 20

  Maximum number of iterations allowed.
- int total_iter = 0

  Total number of iterations made.
- int nl_method = FINCH_Picard

  Non-linear solution method - default = FINCH_Picard.
- std::vector< double > CL_I

  Left side, implicit coefficients (Calculated at Runtime)
- std::vector< double > CL_E

  Left side, explicit coefficients (Calculated at Runtime)
- std::vector< double > CC_I

  Centered, implicit coefficients (Calculated at Runtime)
- std::vector< double > CC_E

  Centered, explicit coefficients (Calculated at Runtime)
- std::vector< double > CR_I

  Right side, implicit coefficients (Calculated at Runtime)
- std::vector< double > CR_E

  Right side, explicit coefficients (Calculated at Runtime)
- std::vector< double > fL_I

  Left side, implicit fluxes (Calculated at Runtime)
- std::vector< double > fL_E

  Left side, explicit fluxes (Calculated at Runtime)
- std::vector< double > fC_I

  Centered, implicit fluxes (Calculated at Runtime)
- std::vector< double > fC_E

  Centered, explicit fluxes (Calculated at Runtime)

- std::vector< double > fR_I

  Right side, implicit fluxes (Calculated at Runtime)
- std::vector< double > fR_E

  Right side, explicit fluxes (Calculated at Runtime)
- std::vector< double > OI

  Implicit upper diagonal matrix elements (Calculated at Runtime)
- std::vector< double > OE

  Explicit upper diagonal matrix elements (Calculated at Runtime)
- std::vector< double > NI

  Implicit diagonal matrix elements (Calculated at Runtime)
- std::vector< double > NE

  Explicit diagonal matrix elements (Calculated at Runtime)
- std::vector< double > MI

  Implicit lower diagonal matrix elements (Calculated at Runtime)
- std::vector< double > ME

  Explicit lower diagonal matrix elements (Calculated at Runtime)
- std::vector< double > uz_I_I
- std::vector< double > uz_Im1_I
- std::vector< double > uz_Ip1_I

  Implicit local slopes (Calculated at Runtime)
- std::vector< double > uz_I_E
- std::vector< double > uz_Im1_E
- std::vector< double > uz_Ip1_E

  Explicit local slopes (Calculated at Runtime)
- Matrix< double > unm1

  Conserved Quantity Older.
- Matrix< double > un

  Conserved Quantity Old.
- Matrix< double > unp1

  Conserved Quantity New.
- Matrix< double > u_star

  Conserved Quantity Projected New.
- Matrix< double > ubest

  Best found solution if solving iteratively.
- Matrix< double > vn

  Velocity Old.
- Matrix< double > vnp1

  Velocity New.
- Matrix< double > Dn

  Dispersion Old.
- Matrix< double > Dnp1

  Dispersion New.
- Matrix< double > kn

  Reaction Old.
- Matrix< double > knp1

  Reaction New.
- Matrix< double > Sn

  Forcing Function Old.
- Matrix< double > Snp1

  Forcing Function New.
- Matrix< double > Rn

Time Coeff Old.

- Matrix< double > Rnp1

  Time Coeff New.

- Matrix< double > Fn

  Flux Limiter Old.

- Matrix< double > Fnp1

  Flux Limiter New.

- Matrix< double > gI

  Implicit Side Boundary Conditions.

- Matrix< double > gE

  Explicit Side Boundary Conditions.

- Matrix< double > res

  Current residual.

- Matrix< double > pres

  Current search direction.

- int(* callroutine)(const void *user_data)

  Function pointer to executioner (DEFAULT = default_execution)

- int(* setic)(const void *user_data)

  Function pointer to initial conditions (DEFAULT = default_ic)

- int(* settime)(const void *user_data)

  Function pointer to set time step (DEFAULT = default_timestep)

- int(* setpreprocess)(const void *user_data)

  Function pointer to preprocesses (DEFAULT = default_preprocess)

- int(* solve)(const void *user_data)

  Function pointer to the solver (DEFAULT = default_solve)

- int(* setparams)(const void *user_data)

  Function pointer to set parameters (DEFAULT = default_params)

- int(* discretize)(const void *user_data)

  Function pointer to discretization (DEFAULT = ospre_discretization)

- int(* setbcs)(const void *user_data)
- int(* evalres)(const Matrix< double > &x, Matrix< double > &res, const void *user_data)

  Function pointer to the residual function (DEFAULT = default_res)

- int(* evalprecon)(const Matrix< double > &b, Matrix< double > &p, const void *user_data)

  Function pointer to the preconditioning function (DEFAULT = default_precon)

- int(* setpostprocess)(const void *user_data)

  Function pointer to the postprocesses (DEFAULT = default_postprocess)

- int(* resettime)(const void *user_data)

  Function pointer to reset time (DEFAULT = default_reset)

- PICARD_DATA picard_dat

  Data structure for PICARD method (no need to use this)

- PJFNK_DATA pjfnk_dat

  Data structure for PJFNK method (more rigours method)

- const void * param_data

  User's data structure used to evaluate the parameter function (Must override if setparams is overriden)

5.30.1 Detailed Description

Data structure for the FINCH object.

C-style object that holds data, functions, and other structures necessary to discretize and solve a FINCH problem. All of this information must be overriden or initialized prior to running a FINCH simulation. Many, many default functions are provided to make it easier to incorporate FINCH into other problems. The main function to override will be the setparams function. This will be a function that the user provides to tell the FINCH simulation how the parameters of the problem vary in time and space and whether or not they are coupled the the variable u. All functions are overridable and several can be skipped entirely, or called directly at different times in the execution of a particular routine. This make FINCH extremely flexible to the user.

Note

All parameters and dimensions do not carry any units with them. The user is required to keep track of all their own units in their particular problem and ensure that units will cancel and be consistent in their own physical model.

Definition at line 81 of file finch.h.

5.30.2 Member Data Documentation

5.30.2.1 double FINCH_DATA::beta = 0.5

Scheme type indicator: 0.5=CN & 1.0=Implicit; all else NULL.

Definition at line 123 of file finch.h.

5.30.2.2 int(* FINCH_DATA::callroutine)(const void *user_data)

Function pointer to executioner (DEFAULT = default_execution)

Definition at line 186 of file finch.h.

5.30.2.3 std::vector<double> FINCH_DATA::CC_E

Centered, explicit coefficients (Calculated at Runtime)

Definition at line 134 of file finch.h.

5.30.2.4 std::vector<double> FINCH_DATA::CC_I

Centered, implicit coefficients (Calculated at Runtime)

Definition at line 133 of file finch.h.

5.30.2.5 bool FINCH_DATA::CheckMass = false

Flag to indicate whether or not mass is to be checked.

Definition at line 117 of file finch.h.

5.30.2.6 std::vector<double> FINCH_DATA::CL_E

Left side, explicit coefficients (Calculated at Runtime)

Definition at line 132 of file finch.h.

5.30.2.7 std::vector<double> FINCH_DATA::CL_I

Left side, implicit coefficients (Calculated at Runtime)

Definition at line 131 of file finch.h.

5.30.2.8 bool FINCH_DATA::CN = true

True if Crank-Nicholson, false if Implicit, never use explicit.

Definition at line 114 of file finch.h.

5.30.2.9 std::vector<double> FINCH_DATA::CR_E

Right side, explicit coefficients (Calculated at Runtime)

Definition at line 136 of file finch.h.

5.30.2.10 std::vector<double> FINCH_DATA::CR_I

Right side, implicit coefficients (Calculated at Runtime)

Definition at line 135 of file finch.h.

5.30.2.11 int FINCH_DATA::d = 0

Dimension of the problem: 0 = cartesian, 1 = cylindrical, 2 = spherical.

Definition at line 84 of file finch.h.

5.30.2.12 double FINCH_DATA::DIC = 1.0

Initial condition of Dispersion (if constant)

Definition at line 100 of file finch.h.

5.30.2.13 bool FINCH_DATA::Dirichlet = false

Flag to indicate use of Dirichlet or Neumann starting boundary.

Definition at line 116 of file finch.h.

5.30.2.14 int(* FINCH_DATA::discretize)(const void *user_data)

Function pointer to discretization (DEFAULT = ospre_discretization)

Definition at line 192 of file finch.h.

5.30.2.15 Matrix<double> FINCH_DATA::Dn

Dispersion Old.

Definition at line 166 of file finch.h.

5.30.2.16 Matrix<double> FINCH_DATA::Dnp1

Dispersion New.

Definition at line 167 of file finch.h.

5.30.2.17 double FINCH_DATA::Do = 1.0

Boundary Value of Dispersion.

Definition at line 105 of file finch.h.

5.30.2.18 double FINCH_DATA::dt = 0.0125

Time step.

Definition at line 85 of file finch.h.

5.30.2.19 double FINCH_DATA::dt_old = 0.0125

Previous time step.

Definition at line 86 of file finch.h.

5.30.2.20 double FINCH_DATA::dz = 0.1

Space step.

Definition at line 88 of file finch.h.

5.30.2.21 int(* FINCH_DATA::evalprecon)(const Matrix< double > &b, Matrix< double > &p, const void *user_data)

Function pointer to the preconditioning function (DEFAULT = default_precon)

Definition at line 197 of file finch.h.

5.30.2.22 int(* FINCH_DATA::evalres)(const Matrix< double > &x, Matrix< double > &res, const void *user_data)

Function pointer to the residual function (DEFAULT = default_res)

Definition at line 195 of file finch.h.

5.30.2.23 bool FINCH_DATA::ExplicitFlux = false

Flag to indicate whether or not to use fully explicit flux limiters.

Definition at line 118 of file finch.h.

5.30.2.24 std::vector<double> FINCH_DATA::fC_E

Centered, explicit fluxes (Calculated at Runtime)

Definition at line 141 of file finch.h.

5.30.2.25 std::vector<double> FINCH_DATA::fC_I

Centered, implicit fluxes (Calculated at Runtime)

Definition at line 140 of file finch.h.

5.30.2.26 std::vector<double> FINCH_DATA::fL_E

Left side, explicit fluxes (Calculated at Runtime)

Definition at line 139 of file finch.h.

5.30.2.27 std::vector<double> FINCH_DATA::fL_I

Left side, implicit fluxes (Calculated at Runtime)

Definition at line 138 of file finch.h.

5.30.2.28 Matrix<double> FINCH_DATA::Fn

Flux Limiter Old.

Definition at line 175 of file finch.h.

5.30.2.29 Matrix<double> FINCH_DATA::Fnp1

Flux Limiter New.

Definition at line 176 of file finch.h.

5.30.2.30 std::vector<double> FINCH_DATA::fR_E

Right side, explicit fluxes (Calculated at Runtime)

Definition at line 143 of file finch.h.

5.30.2.31 std::vector<double> FINCH_DATA::fR_I

Right side, implicit fluxes (Calculated at Runtime)

Definition at line 142 of file finch.h.

5.30.2.32 Matrix<double> FINCH_DATA::gE

Explicit Side Boundary Conditions.

Definition at line 178 of file finch.h.

5.30.2.33 Matrix<double> FINCH_DATA::gI

Implicit Side Boundary Conditions.

Definition at line 177 of file finch.h.

5.30.2.34 bool FINCH_DATA::Iterative = true

Flag to indicate whether to solve directly, or iteratively.

Definition at line 119 of file finch.h.

5.30.2.35 double FINCH_DATA::kfn = 1.0

Film mass transfer coefficient Old.

Definition at line 108 of file finch.h.

5.30.2.36 double FINCH_DATA::kfnp1 = 1.0

Film mass transfer coefficient New.

Definition at line 109 of file finch.h.

5.30.2.37 double FINCH_DATA::kIC = 1.0

Initial condition of Reaction (if constant)

Definition at line 101 of file finch.h.

5.30.2.38 Matrix<double> FINCH_DATA::kn

Reaction Old.

Definition at line 168 of file finch.h.

5.30.2.39 Matrix<double> FINCH_DATA::knp1

Reaction New.

Definition at line 169 of file finch.h.

5.30.2.40 double FINCH_DATA::ko = 1.0

Boundary Value of Reaction.

Definition at line 106 of file finch.h.

5.30.2.41 double FINCH_DATA::L = 1.0

Total space.

Definition at line 89 of file finch.h.

5.30.2.42 double FINCH_DATA::lambda_E

Boundary Coefficient for Explicit Neumann (Calculated at Runtime)

Definition at line 111 of file finch.h.

5.30.2.43 double FINCH_DATA::lambda_I

Boundary Coefficient for Implicit Neumann (Calculated at Runtime)

Definition at line 110 of file finch.h.

5.30.2.44 int FINCH_DATA::LN = 10

Number of nodes.

Definition at line 113 of file finch.h.

5.30.2.45 int FINCH_DATA::max_iter = 20

Maximum number of iterations allowed.

Definition at line 126 of file finch.h.

5.30.2.46 std::vector<double> FINCH_DATA::ME

Explicit lower diagonal matrix elements (Calculated at Runtime)

Definition at line 151 of file finch.h.

5.30.2.47 std::vector<double> FINCH_DATA::MI

Implicit lower diagonal matrix elements (Calculated at Runtime)

Definition at line 150 of file finch.h.

5.30.2.48 std::vector<double> FINCH_DATA::NE

Explicit diagonal matrix elements (Calculated at Runtime)

Definition at line 149 of file finch.h.

5.30.2.49 std::vector<double> FINCH_DATA::NI

Implicit diagonal matrix elements (Calculated at Runtime)

Definition at line 148 of file finch.h.

5.30.2.50 int FINCH_DATA::nl_method = FINCH_Picard

Non-linear solution method - default = FINCH_Picard.

Definition at line 128 of file finch.h.

5.30.2.51 bool FINCH_DATA::NormTrack = true

Flag to determine whether or not to track the norms during simulation.

Definition at line 121 of file finch.h.

5.30.2.52 std::vector<double> FINCH_DATA::OE

Explicit upper diagonal matrix elements (Calculated at Runtime)

Definition at line 147 of file finch.h.

5.30.2.53 std::vector<double> FINCH_DATA::OI

Implicit upper diagonal matrix elements (Calculated at Runtime)

Definition at line 146 of file finch.h.

5.30.2.54 const void* FINCH_DATA::param_data

User's data structure used to evaluate the parameter function (Must override if setparams is overriden)

Definition at line 204 of file finch.h.

5.30.2.55 PICARD_DATA FINCH_DATA::picard_dat

Data structure for PICARD method (no need to use this)

Definition at line 202 of file finch.h.

5.30.2.56 PJFNK_DATA FINCH_DATA::pjfnk_dat

Data structure for PJFNK method (more rigours method)

Definition at line 203 of file finch.h.

5.30.2.57 Matrix<double> FINCH_DATA::pres

Current search direction.

Definition at line 181 of file finch.h.

5.30.2.58 Matrix<double> FINCH_DATA::res

Current residual.

Definition at line 180 of file finch.h.

5.30.2.59 int(* FINCH_DATA::resettime)(const void *user_data)

Function pointer to reset time (DEFAULT = default_reset)

Definition at line 199 of file finch.h.

5.30.2.60 double FINCH_DATA::RIC = 1.0

Initial condition of the Time Coefficient (if constant)

Definition at line 102 of file finch.h.

5.30.2.61 Matrix<double> FINCH_DATA::Rn

Time Coeff Old.

Definition at line 172 of file finch.h.

5.30.2.62 Matrix<double> FINCH_DATA::Rnp1

Time Coeff New.

Definition at line 173 of file finch.h.

5.30.2.63 double FINCH_DATA::Ro = 1.0

Boundary Value of Time Coefficient.

Definition at line 107 of file finch.h.

5.30.2.64 double FINCH_DATA::s = 1.0

Char quantity (spherical = 1, cylindrical = length, cartesian = area)

Definition at line 90 of file finch.h.

5.30.2.65 int(* FINCH_DATA::setbcs)(const void *user_data)

Function pointer to set boundary conditions (DEFAULT = default_bcs)

Definition at line 193 of file finch.h.

5.30.2.66 int(* FINCH_DATA::setic)(const void *user_data)

Function pointer to initial conditions (DEFAULT = default_ic)

Definition at line 187 of file finch.h.

5.30.2.67 int(* FINCH_DATA::setparams)(const void *user_data)

Function pointer to set parameters (DEFAULT = default_params)

Definition at line 191 of file finch.h.

5.30.2.68 int(* FINCH_DATA::setpostprocess)(const void *user_data)

Function pointer to the postprocesses (DEFAULT = default_postprocess)

Definition at line 198 of file finch.h.

5.30.2.69 int(* FINCH_DATA::setpreprocess)(const void *user_data)

Function pointer to preprocesses (DEFAULT = default_preprocess)

Definition at line 189 of file finch.h.

5.30.2.70 int(* FINCH_DATA::settime)(const void *user_data)

Function pointer to set time step (DEFAULT = default_timestep)

Definition at line 188 of file finch.h.

5.30.2.71 Matrix<double> FINCH_DATA::Sn

Forcing Function Old.

Definition at line 170 of file finch.h.

5.30.2.72 Matrix<double> FINCH_DATA::Snp1

Forcing Function New.

Definition at line 171 of file finch.h.

5.30.2.73 int(* FINCH_DATA::solve)(const void *user_data)

Function pointer to the solver (DEFAULT = default_solve)

Definition at line 190 of file finch.h.

5.30.2.74 bool FINCH_DATA::SteadyState = false

Flag to determine whether or not to solve the steady-state problem.

Definition at line 120 of file finch.h.

5.30.2.75 double FINCH_DATA::T = 1.0

Total time.

Definition at line 87 of file finch.h.

5.30.2.76 double FINCH_DATA::t = 0.0

Current Time.

Definition at line 91 of file finch.h.

5.30.2.77 double FINCH_DATA::t_old = 0.0

Previous Time.

Definition at line 92 of file finch.h.

5.30.2.78 double FINCH_DATA::tol_abs = 1e-6

Absolute Tolerance for Convergence.

Definition at line 125 of file finch.h.

5.30.2.79 double FINCH_DATA::tol_rel = 1e-6

Relative Tolerance for Convergence.

Definition at line 124 of file finch.h.

5.30.2.80 int FINCH_DATA::total_iter = 0

Total number of iterations made.

Definition at line 127 of file finch.h.

5.30.2.81 Matrix<double> FINCH_DATA::u_star

Conserved Quantity Projected New.

Definition at line 161 of file finch.h.

5.30.2.82 double FINCH_DATA::uAvg = 0.0

Average amount of conserved quantity in domain.

Definition at line 96 of file finch.h.

5.30.2.83 double FINCH_DATA::uAvg_old = 0.0

Old Average amount of conserved quantity.

Definition at line 97 of file finch.h.

5.30.2.84 Matrix<double> FINCH_DATA::ubest

Best found solution if solving iteratively.

Definition at line 162 of file finch.h.

5.30.2.85 double FINCH_DATA::uIC = 0.0

Initial condition of Conserved Quantity (if constant)

Definition at line 98 of file finch.h.

5.30.2.86 Matrix<double> FINCH_DATA::un

Conserved Quantity Old.

Definition at line 159 of file finch.h.

5.30.2.87 Matrix<double> FINCH_DATA::unm1

Conserved Quantity Older.

Definition at line 158 of file finch.h.

5.30.2.88 Matrix<double> FINCH_DATA::unp1

Conserved Quantity New.

Definition at line 160 of file finch.h.

5.30.2.89 double FINCH_DATA::uo = 1.0

Boundary Value of Conserved Quantity.

Definition at line 103 of file finch.h.

5.30.2.90 bool FINCH_DATA::Update = false

Flag to check if the system needs updating.

Definition at line 115 of file finch.h.

5.30.2.91 double FINCH_DATA::uT = 0.0

Total amount of conserved quantity in domain.

Definition at line 94 of file finch.h.

5.30.2.92 double FINCH_DATA::uT_old = 0.0

Old Total amount of conserved quantity.

Definition at line 95 of file finch.h.

5.30.2.93 std::vector<double> FINCH_DATA::uz_l_E

Definition at line 155 of file finch.h.

5.30.2.94 std::vector<double> FINCH_DATA::uz_l_I

Definition at line 154 of file finch.h.

5.30.2.95 std::vector<double> FINCH_DATA::uz_lm1_E

Definition at line 155 of file finch.h.

5.30.2.96 std::vector<double> FINCH_DATA::uz_lm1_I

Definition at line 154 of file finch.h.

5.30.2.97 std::vector<double> FINCH_DATA::uz_lp1_E

Explicit local slopes (Calculated at Runtime)

Definition at line 155 of file finch.h.

5.30.2.98 std::vector<double> FINCH_DATA::uz_lp1_I

Implicit local slopes (Calculated at Runtime)

Definition at line 154 of file finch.h.

5.30.2.99 double FINCH_DATA::vIC = 1.0

Initial condition of Velocity (if constant)

Definition at line 99 of file finch.h.

5.30.2.100 Matrix<double> FINCH_DATA::vn

Velocity Old.

Definition at line 164 of file finch.h.

5.30.2.101 Matrix<double> FINCH_DATA::vnp1

Velocity New.

Definition at line 165 of file finch.h.

5.30.2.102 double FINCH_DATA::vo = 1.0

Boundary Value of Velocity.

Definition at line 104 of file finch.h.

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

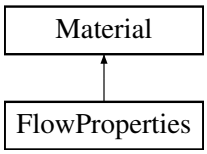
- finch.h

5.31 FlowProperties Class Reference

FlowProperties class object inherits from Material object.

`#include <FlowProperties.h>`

Inheritance diagram for FlowProperties:



Public Member Functions

- FlowProperties (const InputParameters &parameters)
  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual void computeQpProperties ()

Required function override for Material objects in MOOSE.

Private Attributes

- std::vector< Real > _molecular_wieght

  Molecular wieghts for each gas species (g/mol)
- std::vector< Real > _comp_heat_capacity

  Heat capacities for each gas species (J/g/K)
- std::vector< Real > _comp_ref_viscosity

  Sutherland's reference viscosity for each gas species (g/cm/s)
- std::vector< Real > _comp_ref_temp

  Sutherland's reference temperature for each species (K)
- std::vector< Real > _comp_Sutherland_const

  Sutherland's constant for each gas species (K)
- Real _flow_rate

  Inlet flow rate for the fixed-bed column (cm^3/hr)
- Real _column_length

  Length of the fixed-bed column (cm)
- MaterialProperty< Real > & _velocity

  MaterialProperty for the linear velocity in the bed (cm/hr)
- MaterialProperty< Real > & _gas_density

  MaterialProperty for the gas density (g/cm^3)
- MaterialProperty< Real > & _gas_viscosity

  MaterialProperty for the gas viscosity (g/cm/s)
- MaterialProperty< Real > & _gas_heat_capacity

  MaterialProperty for the gas heat capacity (J/g/K)
- MaterialProperty< Real > & _gas_molecular_wieght

  MaterialProperty for the gas total molecular wieght (g/mol)
- const MaterialProperty< Real > & _inner_dia

  Coupled material property for bed inner diameter.
- const MaterialProperty< Real > & _porosity

  Coupled material property for bed bulk porosity.
- const MaterialProperty< Real > & _pellet_density

  Coupled material property for adsorbent pellet density.
- const MaterialProperty< Real > & _pellet_heat_capacity

  Coupled material property for adsorbent heat capacity.
- MaterialProperty< Real > & _heat_retardation

  MaterialProperty for energy balance retardation coefficient.
- MaterialProperty< std::vector < Real > > & _molecular_diffusion

  MaterialProperty for each species' molecular diffusion (cm^2/s)
- MaterialProperty< std::vector < Real > > & _dispersion

  MaterialProperty for each species' dispersion coefficient (cm^2/hr)
- MaterialProperty< std::vector < Real > > & _retardation

  MaterialProperty for each species' retardation coefficient.
- VariableValue & _temperature

  Reference to the coupled column temperature.
- VariableValue & _total_pressure

  Reference to the coupled column pressure.

- std::vector< unsigned int > _index

  Indices for the gas species in the system.
- std::vector< VariableValue * > _gas_conc

  Pointer list to the coupled gases.
- std::vector< VariableValue * > _solid_conc

  Pointer list to the coupled adsorption concentrations.
- std::vector< VariableValue * > _solid_perturb

  Pointer list to the coupled adsorption perturbations.

5.31.1 Detailed Description

FlowProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up and calculate various flow properties including linear velocities, molecular diffusion, mechanical dispersion, gas density, gas viscosity, gas heat capacity, etc. This object also approximates the effective retardation coefficient for each species in the mass balance. The evaluation of that parameter is dependent on the solid phase concentration variable, which will either be calculated by MAGPIE (see magpie.h) or SCOPSOWL (see scopsowl.h) dependending on whether or not we will consider adsorption kinetics in the simulation.

Definition at line 63 of file FlowProperties.h.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 FlowProperties::FlowProperties (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.31.3 Member Function Documentation

5.31.3.1 virtual void FlowProperties::computeQpProperties ( ) [protected], [virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

5.31.4 Member Data Documentation

5.31.4.1 Real FlowProperties::_column_length [private]

Length of the fixed-bed column (cm)

Definition at line 81 of file FlowProperties.h.

5.31.4.2 std::vector<Real> FlowProperties::_comp_heat_capacity [private]

Heat capacities for each gas species (J/g/K)

Definition at line 76 of file FlowProperties.h.

5.31.4.3 std::vector<Real> FlowProperties::_comp_ref_temp [private]

Sutherland's reference temperature for each species (K)

Definition at line 78 of file FlowProperties.h.

5.31.4.4 std::vector<Real> FlowProperties::_comp_ref_viscosity [private]

Sutherland's reference viscosity for each gas species (g/cm/s)

Definition at line 77 of file FlowProperties.h.

5.31.4.5 std::vector<Real> FlowProperties::_comp_Sutherland_const [private]

Sutherland's constant for each gas species (K)

Definition at line 79 of file FlowProperties.h.

5.31.4.6 MaterialProperty<std::vector<Real> >& FlowProperties::_dispersion [private]

MaterialProperty for each species' dispersion coefficient (cm^2/hr)

Definition at line 96 of file FlowProperties.h.

5.31.4.7 Real FlowProperties::_flow_rate [private]

Inlet flow rate for the fixed-bed column (cm^3/hr)

Definition at line 80 of file FlowProperties.h.

5.31.4.8 std::vector<VariableValue *> FlowProperties::_gas_conc [private]

Pointer list to the coupled gases.

Definition at line 102 of file FlowProperties.h.

5.31.4.9 MaterialProperty<Real>& FlowProperties::_gas_density [private]

MaterialProperty for the gas density (g/cm^3)

Definition at line 84 of file FlowProperties.h.

5.31.4.10 MaterialProperty<Real>& FlowProperties::_gas_heat_capacity [private]

MaterialProperty for the gas heat capacity (J/g/K)

Definition at line 86 of file FlowProperties.h.

5.31.4.11 MaterialProperty<Real>& FlowProperties::_gas_molecular_wieght [private]

MaterialProperty for the gas total molecular wieght (g/mol)

Definition at line 87 of file FlowProperties.h.

5.31.4.12 MaterialProperty<Real>& FlowProperties::_gas_viscosity [private]

MaterialProperty for the gas viscosity (g/cm/s)

Definition at line 85 of file FlowProperties.h.

5.31.4.13 MaterialProperty<Real>& FlowProperties::_heat_retardation [private]

MaterialProperty for energy balance retardation coefficient.

Definition at line 94 of file FlowProperties.h.

5.31.4.14 std::vector<unsigned int> FlowProperties::_index [private]

Indices for the gas species in the system.

Definition at line 101 of file FlowProperties.h.

5.31.4.15 const MaterialProperty<Real>& FlowProperties::_inner_dia [private]

Coupled material property for bed inner diameter.

Definition at line 89 of file FlowProperties.h.

5.31.4.16 MaterialProperty<std::vector<Real> >& FlowProperties::_molecular_diffusion [private]

MaterialProperty for each species' molecular diffusion (cm^2/s)

Definition at line 95 of file FlowProperties.h.

5.31.4.17 std::vector<Real> FlowProperties::_molecular_wieght [private]

Molecular wieghts for each gas species (g/mol)

Definition at line 75 of file FlowProperties.h.

5.31.4.18 const MaterialProperty<Real>& FlowProperties::_pellet_density [private]

Coupled material property for adsorbent pellet density.

Definition at line 91 of file FlowProperties.h.

5.31.4.19 const MaterialProperty<Real>& FlowProperties::_pellet_heat_capacity [private]

Coupled material property for adsorbent heat capacity.

Definition at line 92 of file FlowProperties.h.

5.31.4.20 const MaterialProperty<Real>& FlowProperties::_porosity [private]

Coupled material property for bed bulk porosity.

Definition at line 90 of file FlowProperties.h.

5.31.4.21 MaterialProperty<std::vector<Real> >& FlowProperties::_retardation [private]

MaterialProperty for each species' retardation coefficient.

Definition at line 97 of file FlowProperties.h.

5.31.4.22 std::vector<VariableValue *> FlowProperties::_solid_conc [private]

Pointer list to the coupled adsorption concentrations.

Definition at line 103 of file FlowProperties.h.

5.31.4.23 std::vector<VariableValue *> FlowProperties::_solid_perturb [private]

Pointer list to the coupled adsorption perturbations.

Definition at line 104 of file FlowProperties.h.

5.31.4.24 VariableValue& FlowProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 99 of file FlowProperties.h.

5.31.4.25 VariableValue& FlowProperties::_total_pressure [private]

Reference to the coupled column pressure.

Definition at line 100 of file FlowProperties.h.

5.31.4.26 MaterialProperty<Real>& FlowProperties::_velocity [private]

MaterialProperty for the linear velocity in the bed (cm/hr)

Definition at line 83 of file FlowProperties.h.

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

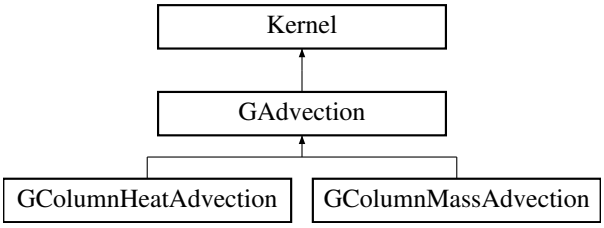
- FlowProperties.h

5.32 GAdvection Class Reference

GAdvection class object inherits from Kernel object.

`#include <GAdvection.h>`

Inheritance diagram for GAdvection:



Public Member Functions

- GAdvection (const InputParameters &parameters)
  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()
  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()
  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- RealVectorValue _velocity
  Vector of velocity.
- Real _vx
  x-component of velocity (optional - set in input file)
- Real _vy
  y-component of velocity (optional - set in input file)
- Real _vz
  z-component of velocity (optional - set in input file)

5.32.1 Detailed Description

GAdvection class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel has a velocity vector whose components can be set piecewise in an input file.

Note

To create a specific GAdvection kernel, inherit from this class and override the components of the velocity vector, then call the residual and Jacobian functions for this object.

Definition at line 58 of file GAdvection.h.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 GAdvection::GAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.32.3 Member Function Documentation

5.32.3.1 virtual Real GAdvection::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in GColumnHeatAdvection, and GColumnMassAdvection.

5.32.3.2 virtual Real GAdvection::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in GColumnHeatAdvection, and GColumnMassAdvection.

5.32.4 Member Data Documentation

5.32.4.1 RealVectorValue GAdvection::_velocity [protected]

Vector of velocity.

Definition at line 74 of file GAdvection.h.

5.32.4.2 Real GAdvection::_vx [protected]

x-component of velocity (optional - set in input file)

Definition at line 76 of file GAdvection.h.

5.32.4.3 Real GAdvection::_vy [protected]

y-component of velocity (optional - set in input file)

Definition at line 77 of file GAdvection.h.

5.32.4.4 Real GAdvection::_vz [protected]

z-component of velocity (optional - set in input file)

Definition at line 78 of file GAdvection.h.

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

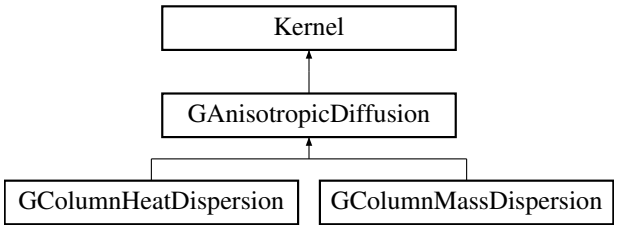
- GAdvection.h

5.33 GAnisotropicDiffusion Class Reference

GAnisotropicDiffusion class object inherits from Kernel object.

`#include <GAnisotropicDiffusion.h>`

Inheritance diagram for GAnisotropicDiffusion:



Public Member Functions

- GAnisotropicDiffusion (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- RealTensorValue _Diffusion

  Diffusion tensor matrix parameter.
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz

5.33.1 Detailed Description

GAnisotropicDiffusion class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel has a diffusion tensor whose components can be set piecewise in an input file.

Note

To create a specific GAnisotropicDiffusion kernel, inherit from this class and override the components of the diffusion tensor, then call the residual and Jacobian functions for this object.

Definition at line 58 of file GAnisotropicDiffusion.h.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 GAnisotropicDiffusion::GAnisotropicDiffusion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.33.3 Member Function Documentation

5.33.3.1 virtual Real GAnisotropicDiffusion::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in GColumnHeatDispersion, and GColumnMassDispersion.

5.33.3.2 virtual Real GAnisotropicDiffusion::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in GColumnHeatDispersion, and GColumnMassDispersion.

5.33.4 Member Data Documentation

5.33.4.1 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected]

Diffusion tensor matrix parameter.

Definition at line 74 of file GAnisotropicDiffusion.h.

5.33.4.2 Real GAnisotropicDiffusion::_Dxx [protected]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.33.4.3 Real GAnisotropicDiffusion::_Dxy [protected]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.33.4.4 Real GAnisotropicDiffusion::_Dxz [protected]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.33.4.5 Real GAnisotropicDiffusion::_Dyx [protected]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.33.4.6 Real GAnisotropicDiffusion::_Dyy [protected]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.33.4.7 Real GAnisotropicDiffusion::_Dyz [protected]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.33.4.8 Real GAnisotropicDiffusion::_Dzx [protected]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.33.4.9 Real GAnisotropicDiffusion::_Dzy [protected]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.33.4.10 Real GAnisotropicDiffusion::_Dzz [protected]

Definition at line 78 of file GAnisotropicDiffusion.h.

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

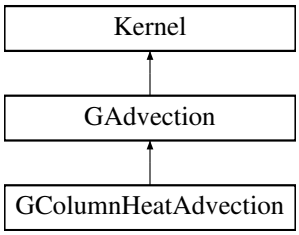
- GAnisotropicDiffusion.h

5.34 GColumnHeatAdvection Class Reference

GColumnHeatAdvection class object inherits from GAdvection object.

`#include <GColumnHeatAdvection.h>`

Inheritance diagram for GColumnHeatAdvection:



Public Member Functions

- GColumnHeatAdvection (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Vector of velocity.
- Real _vx

  x-component of velocity (optional - set in input file)
- Real _vy

  y-component of velocity (optional - set in input file)
- Real _vz

  z-component of velocity (optional - set in input file)

Private Attributes

- const MaterialProperty< Real > & _vel

  Reference to the linear velocity material property.
- const MaterialProperty< Real > & _gas_density

  Reference to the gas density material property.
- const MaterialProperty< Real > & _gas_heat_capacity

  Reference to the gas heat capacity material property.

5.34.1 Detailed Description

GColumnHeatAdvection class object inherits from GAdvection object.

This class object inherits from the generic GAdvection kernel for use with the corresponding DGColumnHeatAdvection kernel to complete the physical description of DG methods in MOOSE. It is coupled with the material properties of linear velocity, gas density, and gas heat capacity and uses those parameters to override the components of the velocity vector of the more generic GAdvection class.

Definition at line 57 of file GColumnHeatAdvection.h.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 GColumnHeatAdvection::GColumnHeatAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.34.3 Member Function Documentation

5.34.3.1 virtual Real GColumnHeatAdvection::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from GAdvection.

5.34.3.2 virtual Real GColumnHeatAdvection::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from GAdvection.

5.34.4 Member Data Documentation

5.34.4.1 const MaterialProperty<Real>& GColumnHeatAdvection::_gas_density [private]

Reference to the gas density material property.

Definition at line 75 of file GColumnHeatAdvection.h.

5.34.4.2 const MaterialProperty<Real>& GColumnHeatAdvection::_gas_heat_capacity [private]

Reference to the gas heat capacity material property.

Definition at line 76 of file GColumnHeatAdvection.h.

5.34.4.3 const MaterialProperty<Real>& GColumnHeatAdvection::_vel [private]

Reference to the linear velocity material property.

Definition at line 74 of file GColumnHeatAdvection.h.

5.34.4.4 RealVectorValue GAdvection::_velocity [protected], [inherited]

Vector of velocity.

Definition at line 74 of file GAdvection.h.

5.34.4.5 Real GAdvection::_vx [protected], [inherited]

x-component of velocity (optional - set in input file)

Definition at line 76 of file GAdvection.h.

5.34.4.6 Real GAdvection::_vy [protected], [inherited]

y-component of velocity (optional - set in input file)

Definition at line 77 of file GAdvection.h.

5.34.4.7 Real GAdvection::_vz [protected], [inherited]

z-component of velocity (optional - set in input file)

Definition at line 78 of file GAdvection.h.

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

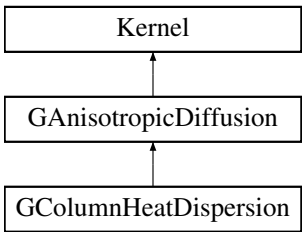
- GColumnHeatAdvection.h

5.35 GColumnHeatDispersion Class Reference

GColumnHeatDispersion class object inherits from GAnisotropicDiffusion object.

`#include <GColumnHeatDispersion.h>`

Inheritance diagram for GColumnHeatDispersion:



Public Member Functions

- GColumnHeatDispersion (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- RealTensorValue _Diffusion

  Diffusion tensor matrix parameter.
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz

Private Attributes

- const MaterialProperty< Real > & _conductivity

  Reference to the thermal conductivity material property.

5.35.1 Detailed Description

GColumnHeatDispersion class object inherits from GAnisotropicDiffusion object.

This class object inherits from the GAnisotropicDiffusion object in DGOSPREY. It must be used in conjunction with the DGColumnHeatDispersion object to complete the physical description of diffusion for DG methods in MOOSE. The conductivity material property is coupled with this object and is used to form/override the diffusion tensor of the base class. Then the base class methods are called to form the residuals and Jacobian elements.

Definition at line 56 of file GColumnHeatDispersion.h.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 GColumnHeatDispersion::GColumnHeatDispersion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.35.3 Member Function Documentation

5.35.3.1 virtual Real GColumnHeatDispersion::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from GAnisotropicDiffusion.

5.35.3.2 virtual Real GColumnHeatDispersion::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from GAnisotropicDiffusion.

5.35.4 Member Data Documentation

5.35.4.1 const MaterialProperty<Real>& GColumnHeatDispersion::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 73 of file GColumnHeatDispersion.h.

5.35.4.2 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 74 of file GAnisotropicDiffusion.h.

5.35.4.3 Real GAnisotropicDiffusion::_Dxx [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.35.4.4 Real GAnisotropicDiffusion::_Dxy [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.35.4.5 Real GAnisotropicDiffusion::_Dxz [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.35.4.6 Real GAnisotropicDiffusion::_Dyx [protected], [inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.35.4.7 Real GAnisotropicDiffusion::_Dyy [protected], [inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.35.4.8 Real GAnisotropicDiffusion::_Dyz [protected], [inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.35.4.9 Real GAnisotropicDiffusion::_Dzx [protected], [inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.35.4.10 Real GAnisotropicDiffusion::_Dzy [protected], [inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.35.4.11 Real GAnisotropicDiffusion::_Dzz [protected], [inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

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

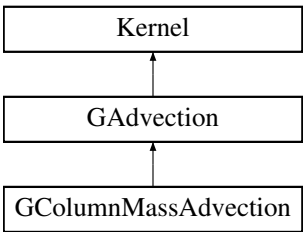
- GColumnHeatDispersion.h

5.36 GColumnMassAdvection Class Reference

GColumnMassAdvection class object inherits from GAdvection object.

`#include <GColumnMassAdvection.h>`

Inheritance diagram for GColumnMassAdvection:



Public Member Functions

- GColumnMassAdvection (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- RealVectorValue _velocity

  Vector of velocity.
- Real _vx

  x-component of velocity (optional - set in input file)
- Real _vy

  y-component of velocity (optional - set in input file)
- Real _vz

  z-component of velocity (optional - set in input file)

Private Attributes

- const MaterialProperty< Real > & _vel

  Reference to the linear velocity material property.

5.36.1 Detailed Description

GColumnMassAdvection class object inherits from GAdvection object.

This class object inherits from the generic GAdvection kernel for use with the corresponding DGColumnMassAdvection kernel to complete the physical description of DG methods in MOOSE. It is coupled with the material property of linear velocity and uses that parameter to override the components of the velocity vector of the more generic GAdvection class.

Definition at line 55 of file GColumnMassAdvection.h.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 GColumnMassAdvection::GColumnMassAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.36.3 Member Function Documentation

5.36.3.1 virtual Real GColumnMassAdvection::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from GAdvection.

5.36.3.2 virtual Real GColumnMassAdvection::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from GAdvection.

5.36.4 Member Data Documentation

5.36.4.1 const MaterialProperty<Real>& GColumnMassAdvection::_vel [private]

Reference to the linear velocity material property.

Definition at line 72 of file GColumnMassAdvection.h.

5.36.4.2 RealVectorValue GAdvection::_velocity [protected], [inherited]

Vector of velocity.

Definition at line 74 of file GAdvection.h.

5.36.4.3 Real GAdvection::_vx [protected], [inherited]

x-component of velocity (optional - set in input file)

Definition at line 76 of file GAdvection.h.

5.36.4.4 Real GAdvection::_vy [protected], [inherited]

y-component of velocity (optional - set in input file)

Definition at line 77 of file GAdvection.h.

5.36.4.5 Real GAdvection::_vz [protected], [inherited]

z-component of velocity (optional - set in input file)

Definition at line 78 of file GAdvection.h.

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

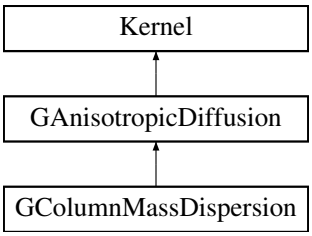
- GColumnMassAdvection.h

5.37 GColumnMassDispersion Class Reference

GColumnMassDispersion class object inherits from GAnisotropicDiffusion object.

`#include <GColumnMassDispersion.h>`

Inheritance diagram for GColumnMassDispersion:



Public Member Functions

- GColumnMassDispersion (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- RealTensorValue _Diffusion

  Diffusion tensor matrix parameter.
- Real _Dxx
- Real _Dxy
- Real _Dxz
- Real _Dyx
- Real _Dyy
- Real _Dyz
- Real _Dzx
- Real _Dzy
- Real _Dzz

Private Attributes

- unsigned int _index

  Index of the species of interest for this kernel.
- const MaterialProperty
  < std::vector< Real > > & _dispersion

  Reference to the dispersion material property.
- const MaterialProperty
  < std::vector< Real > > & _molecular_diffusion

  Reference to the molecular diffusion material property.

5.37.1 Detailed Description

GColumnMassDispersion class object inherits from GAnisotropicDiffusion object.

This class object inherits from the GAnisotropicDiffusion object in DGOSPREY. It must be used in conjunction with the DGColumnMassDispersion object to complete the physical description of diffusion for DG methods in MOOSE. The dispersion and molecular diffusion material properties are coupled with this object and is used to form/override the diffusion tensor of the base class. Then the base class methods are called to form the residuals and Jacobian elements.

Definition at line 56 of file GColumnMassDispersion.h.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 GColumnMassDispersion::GColumnMassDispersion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.37.3 Member Function Documentation

5.37.3.1 virtual Real GColumnMassDispersion::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from GAnisotropicDiffusion.

5.37.3.2 virtual Real GColumnMassDispersion::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from GAnisotropicDiffusion.

5.37.4 Member Data Documentation

5.37.4.1 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 74 of file GAnisotropicDiffusion.h.

5.37.4.2 const MaterialProperty<std::vector<Real> >& GColumnMassDispersion::_dispersion [private]

Reference to the dispersion material property.

Definition at line 74 of file GColumnMassDispersion.h.

5.37.4.3 Real GAnisotropicDiffusion::_Dxx [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.37.4.4 Real GAnisotropicDiffusion::_Dxy [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.37.4.5 Real GAnisotropicDiffusion::_Dxz [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.37.4.6 Real GAnisotropicDiffusion::_Dyx [protected],[inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.37.4.7 Real GAnisotropicDiffusion::_Dyy [protected],[inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.37.4.8 Real GAnisotropicDiffusion::_Dyz [protected],[inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.37.4.9 Real GAnisotropicDiffusion::_Dzx [protected],[inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.37.4.10 Real GAnisotropicDiffusion::_Dzy [protected],[inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.37.4.11 Real GAnisotropicDiffusion::_Dzz [protected],[inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.37.4.12 unsigned int GColumnMassDispersion::_index [private]

Index of the species of interest for this kernel.

Definition at line 73 of file GColumnMassDispersion.h.

5.37.4.13 const MaterialProperty<std::vector<Real> >& GColumnMassDispersion::_molecular_diffusion [private]

Reference to the molecular diffusion material property.

Definition at line 75 of file GColumnMassDispersion.h.

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

- GColumnMassDispersion.h

5.38 GCR_DATA Struct Reference

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

`#include <lark.h>`

Public Attributes

- int restart = -1

  Restart parameter for outer iterations - default = 20.
- int maxit = 0

  Maximum allowable outer iterations.
- int iter_outer = 0

  Number of outer iterations taken.
- int iter_inner = 0

  Number of inner iterations taken.
- int total_iter = 0

  Total number of iterations taken.
- bool breakdown = false

Boolean to determine if a step has failed.

- double alpha

  Inner iteration step size.
- double beta

  Outer iteration step size.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolute tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm for linear system.
- double relres

  Relative residual norm for linear system.
- double relres_base

  Initial residual norm of the linear system.
- double bestres

  Best found residual norm of the linear system.
- bool Output = true

  True = print messages to the console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual Vector.
- Matrix< double > c_temp

  Temporary c vector to be updated.
- Matrix< double > u_temp

  Temporary u vector to be updated.
- std::vector< Matrix< double > > u

  Vector span for updating x.
- std::vector< Matrix< double > > c

  Vector span for updating r.
- OPTRANS_DATA transpose_dat

  Data structure for Operator Transposition.

5.38.1 Detailed Description

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

C-style object used in conjunction with the Generalized Conjugate Residual (GCR) algorithm for solving a non-symmetric linear system of equations. When the linear system in question has a positive-definite-symmetric component to it, then this algorithm is equivalent to GMRESRP. However, it is generally less efficient than GMRESRP and can suffer breakdowns.

Definition at line 336 of file lark.h.

5.38.2 Member Data Documentation

5.38.2.1 double GCR_DATA::alpha

Inner iteration step size.

Definition at line 345 of file lark.h.

5.38.2.2 double GCR_DATA::bestres

Best found residual norm of the linear system.

Definition at line 352 of file lark.h.

5.38.2.3 Matrix<double> GCR_DATA::bestx

Best found solution to the linear system.

Definition at line 357 of file lark.h.

5.38.2.4 double GCR_DATA::beta

Outer iteration step size.

Definition at line 346 of file lark.h.

5.38.2.5 bool GCR_DATA::breakdown = false

Boolean to determine if a step has failed.

Definition at line 343 of file lark.h.

5.38.2.6 std::vector<Matrix<double> > GCR_DATA::c

Vector span for updating r.

Definition at line 362 of file lark.h.

5.38.2.7 Matrix<double> GCR_DATA::c_temp

Temporary c vector to be updated.

Definition at line 359 of file lark.h.

5.38.2.8 int GCR_DATA::iter_inner = 0

Number of inner iterations taken.

Definition at line 341 of file lark.h.

5.38.2.9 int GCR_DATA::iter_outer = 0

Number of outer iterations taken.

Definition at line 340 of file lark.h.

5.38.2.10 int GCR_DATA::maxit = 0

Maximum allowable outer iterations.

Definition at line 339 of file lark.h.

5.38.2.11 bool GCR_DATA::Output = true

True = print messages to the console.

Definition at line 354 of file lark.h.

5.38.2.12 Matrix<double> GCR_DATA::r

Residual Vector.

Definition at line 358 of file lark.h.

5.38.2.13 double GCR_DATA::relres

Relative residual norm for linear system.

Definition at line 350 of file lark.h.

5.38.2.14 double GCR_DATA::relres_base

Initial residual norm of the linear system.

Definition at line 351 of file lark.h.

5.38.2.15 double GCR_DATA::res

Absolute residual norm for linear system.

Definition at line 349 of file lark.h.

5.38.2.16 int GCR_DATA::restart = -1

Restart parameter for outer iterations - default = 20.

Definition at line 338 of file lark.h.

5.38.2.17 double GCR_DATA::tol_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

Definition at line 348 of file lark.h.

5.38.2.18 double GCR_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 347 of file lark.h.

5.38.2.19 int GCR_DATA::total_iter = 0

Total number of iterations taken.

Definition at line 342 of file lark.h.

5.38.2.20 OPTRANS_DATA GCR_DATA::transpose_dat

Data structure for Operator Transposition.

Definition at line 364 of file lark.h.

5.38.2.21 std::vector<Matrix<double> > GCR_DATA::u

Vector span for updating x.

Definition at line 361 of file lark.h.

5.38.2.22 Matrix<double> GCR_DATA::u_temp

Temporary u vector to be updated.

Definition at line 360 of file lark.h.

5.38.2.23 Matrix<double> GCR_DATA::x

Current solution to the linear system.

Definition at line 356 of file lark.h.

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

- lark.h

5.39 GMRESLP_DATA Struct Reference

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

`#include <lark.h>`

Public Attributes

- int restart = -1

  Restart parameter - default = min(vector_size,20)
- int maxit = 0

  Maximum allowable iterations - default = min(vector_size,1000)
- int iter = 0

  Number of iterations needed for convergence.
- int steps = 0

  Total number of gmres iterations and krylov iterations.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Absolution redisual norm of the linear system.
- double relres

  Relative residual norm of the linear system.
- double relres_base

  Initial residual norm of the linear system.
- double bestres

  Best found residual norm of the linear system.
- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual vector for the linear system.
- ARNOLDI_DATA arnoldi_dat

  Data structure for the kyrlov subspace.

5.39.1 Detailed Description

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

C-style object used in conjunction with Generalized Minimum RESidual Left-Precondtioned (GMRESLP) and Full Orthogonalization Method (FOM) algorithms to iteratively or directly solve a linear system of equations. When using with GMRESLP, you can only check/observe the linear residuals before a restart or after the Arnoldi space is constructed. This is because this object uses Left-side Preconditioning. A faster routine may be GMRESRP, which is able to construct residuals after each Arnoldi iteration.

Definition at line 147 of file lark.h.

5.39.2 Member Data Documentation

5.39.2.1 ARNOLDI_DATA GMRESLP_DATA::arnoldi_dat

Data structure for the kyrlov subspace.

Definition at line 167 of file lark.h.

5.39.2.2 double GMRESLP_DATA::bestres

Best found residual norm of the linear system.

Definition at line 159 of file lark.h.

5.39.2.3 Matrix<double> GMRESLP_DATA::bestx

Best found solution to the linear system.

Definition at line 164 of file lark.h.

5.39.2.4 int GMRESLP_DATA::iter = 0

Number of iterations needed for convergence.

Definition at line 151 of file lark.h.

5.39.2.5 int GMRESLP_DATA::maxit = 0

Maximum allowable iterations - default = min(vector_size,1000)

Definition at line 150 of file lark.h.

5.39.2.6 bool GMRESLP_DATA::Output = true

True = print messages to console.

Definition at line 161 of file lark.h.

5.39.2.7 Matrix<double> GMRESLP_DATA::r

Residual vector for the linear system.

Definition at line 165 of file lark.h.

5.39.2.8 double GMRESLP_DATA::relres

Relative residual norm of the linear system.

Definition at line 157 of file lark.h.

5.39.2.9 double GMRESLP_DATA::relres_base

Initial residual norm of the linear system.

Definition at line 158 of file lark.h.

5.39.2.10 double GMRESLP_DATA::res

Absolution redisual norm of the linear system.

Definition at line 156 of file lark.h.

5.39.2.11 int GMRESLP_DATA::restart = -1

Restart parameter - default = min(vector_size,20)

Definition at line 149 of file lark.h.

5.39.2.12 int GMRESLP_DATA::steps = 0

Total number of gmres iterations and krylov iterations.

Definition at line 152 of file lark.h.

5.39.2.13 double GMRESLP_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 155 of file lark.h.

5.39.2.14 double GMRESLP_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 154 of file lark.h.

5.39.2.15 Matrix<double> GMRESLP_DATA::x

Current solution to the linear system.

Definition at line 163 of file lark.h.

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

- lark.h

5.40 GMRESR_DATA Struct Reference

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

`#include <lark.h>`

Public Attributes

- int gcr_restart = -1
  Number of GCR restarts (default = 20, max = N)
- int gcr_maxit = 0
  Number of GCR iterations.
- int gmres_restart = -1
  Number of GMRES restarts (max = 20)
- int gmres_maxit = 1
  Number of GMRES iterations (max = 5, default = 1)
- int N
  Dimension of the linear system.
- int total_iter
  Total GMRES and GCR iterations.
- int iter_outer
  Total GCR iterations.
- int iter_inner
  Total GMRES iterations.
- bool GCR_Output = true
  True = print GCR messages.
- bool GMRES_Output = false
  True = print GMRES messages.
- double gmres_tol = 0.1

Tolerance relative to GCR iterations.

- double gcr_rel_tol = 1e-6

  Relative outer residual tolerance.

- double gcr_abs_tol = 1e-6

  Absolute outer residual tolerance.

- Matrix< double > arg

  Argument matrix passed between preconditioner and iterator.

- GCR_DATA gcr_dat

  Data structure for the outer GCR steps.

- GMRESRP_DATA gmres_dat

  Data structure for the inner GMRES steps.

- int(* matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *matvec_data)

  User supplied matrix-vector product function.

- int(* terminal_precon)(const Matrix< double > &r, Matrix< double > &p, const void *precon_data)

  Optional user supplied terminal preconditioner.

- const void * matvec_data

  Data structure for the user's matvec function.

- const void * term_precon

  Data structure for the user's terminal preconditioner.

5.40.1 Detailed Description

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

C-style object to be used in conjunction with the Generalized Minimum RESidual Recurive (GMRESR) algorithm. Although the name suggests that this method used GMRES recursively, what it is actually doing is nesting GMRESRP iterations inside the GCR method to form a preconditioner for GCR. The name GMRESR came from literature (Vorst and Vuik, "GMRESR: A family of nested GMRES methods", 1991).

Definition at line 373 of file lark.h.

5.40.2 Member Data Documentation

5.40.2.1 Matrix<double> GMRESR_DATA::arg

Argument matrix passed between preconditioner and iterator.

Definition at line 391 of file lark.h.

5.40.2.2 double GMRESR_DATA::gcr_abs_tol = 1e-6

Absolute outer residual tolerance.

Definition at line 389 of file lark.h.

5.40.2.3 GCR_DATA GMRESR_DATA::gcr_dat

Data structure for the outer GCR steps.

Definition at line 393 of file lark.h.

5.40.2.4 int GMRESR_DATA::gcr_maxit = 0

Number of GCR iterations.

Definition at line 376 of file lark.h.

5.40.2.5 bool GMRESR_DATA::GCR_Output = true

True = print GCR messages.

Definition at line 384 of file lark.h.

5.40.2.6 double GMRESR_DATA::gcr_rel_tol = 1e-6

Relative outer residual tolerance.

Definition at line 388 of file lark.h.

5.40.2.7 int GMRESR_DATA::gcr_restart = -1

Number of GCR restarts (default = 20, max = N)

Definition at line 375 of file lark.h.

5.40.2.8 GMRESRP_DATA GMRESR_DATA::gmres_dat

Data structure for the inner GMRES steps.

Definition at line 394 of file lark.h.

5.40.2.9 int GMRESR_DATA::gmres_maxit = 1

Number of GMRES iterations (max = 5, default = 1)

Definition at line 378 of file lark.h.

5.40.2.10 bool GMRESR_DATA::GMRES_Output = false

True = print GMRES messages.

Definition at line 385 of file lark.h.

5.40.2.11 int GMRESR_DATA::gmres_restart = -1

Number of GMRES restarts (max = 20)

Definition at line 377 of file lark.h.

5.40.2.12 double GMRESR_DATA::gmres_tol = 0.1

Tolerance relative to GCR iterations.

Definition at line 387 of file lark.h.

5.40.2.13 int GMRESR_DATA::iter_inner

Total GMRES iterations.

Definition at line 382 of file lark.h.

5.40.2.14 int GMRESR_DATA::iter_outer

Total GCR iterations.

Definition at line 381 of file lark.h.

5.40.2.15 int(* GMRESR_DATA::matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *matvec_data)

User supplied matrix-vector product function.

Definition at line 397 of file lark.h.

5.40.2.16 const void* GMRESR_DATA::matvec_data

Data structure for the user's matvec function.

Definition at line 401 of file lark.h.

5.40.2.17 int GMRESR_DATA::N

Dimension of the linear system.

Definition at line 379 of file lark.h.

5.40.2.18 const void* GMRESR_DATA::term_precon

Data structure for the user's terminal preconditioner.

Definition at line 402 of file lark.h.

5.40.2.19 int(* GMRESR_DATA::terminal_precon)(const Matrix< double > &r, Matrix< double > &p, const void *precon_data)

Optional user supplied terminal preconditioner.

Definition at line 399 of file lark.h.

5.40.2.20 int GMRESR_DATA::total_iter

Total GMRES and GCR iterations.

Definition at line 380 of file lark.h.

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

- lark.h

5.41 GMRESRP_DATA Struct Reference

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

`#include <lark.h>`

Public Attributes

- int restart = -1
  Restart parameter - default = min(20,vector_size)
- int maxit = 0
  Maximum allowable outer iterations.
- int iter_outer = 0
  Total number of outer iterations.
- int iter_inner = 0
  Total number of inner iterations.
- int iter_total = 0
  Total number of overall iterations.
- double tol_rel = 1e-6
  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6
  Absolute tolerance for convergence - default = 1e-6.
- double res
  Absolute residual norm for linear system.

- double relres
  Relative residual norm for linear system.
- double relres_base
  Initial residual norm of the linear system.
- double bestres
  Best found residual norm of the linear system.
- bool Output = true
  True = print messages to console.
- Matrix< double > x
  Current solution to the linear system.
- Matrix< double > bestx
  Best found solution to the linear system.
- Matrix< double > r
  Residual vector for the linear system.
- std::vector< Matrix< double > > Vk
  (N x k) orthonormal vector basis
- std::vector< Matrix< double > > Zk
  (N x k) preconditioned vector set
- std::vector< std::vector < double > > H
  (k+1 x k) upper Hessenberg storage matrix
- std::vector< std::vector < double > > H_bar
  (k+1 x k) Factorized matrix
- std::vector< double > y
  (k x 1) Vector search direction
- std::vector< double > e0
  (k+1 x 1) Normalized vector with residual info
- std::vector< double > e0_bar
  (k+1 x 1) Factorized normal vector
- Matrix< double > w
  (N) x (1) interim result of the matrix_vector multiplication
- Matrix< double > v
  (N) x (1) holding cell for the column entries of Vk and other interims
- Matrix< double > sum
  (N) x (1) running sum of subspace vectors for use in altering w

5.41.1 Detailed Description

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

C-style object used in conjunction with Generalized Minimum RESidual Right Preconditioned (GMRESRP) algorithm to iteratively solve a linear system of equations. Unlike GMRESLP, the GMRESRP method is capable of checking linear residuals at both the inner and outer steps. As a result, this algorithm may terminate earlier than GMRESLP if it has found a suitable solution during one of the inner steps.

Definition at line 177 of file lark.h.

5.41.2 Member Data Documentation

5.41.2.1 double GMRESRP_DATA::bestres

Best found residual norm of the linear system.

Definition at line 190 of file lark.h.

5.41.2.2 Matrix<double> GMRESRP_DATA::bestx

Best found solution to the linear system.

Definition at line 195 of file lark.h.

5.41.2.3 std::vector< double > GMRESRP_DATA::e0

(k+1 x 1) Normalized vector with residual info

Definition at line 203 of file lark.h.

5.41.2.4 std::vector< double > GMRESRP_DATA::e0_bar

(k+1 x 1) Factorized normal vector

Definition at line 204 of file lark.h.

5.41.2.5 std::vector< std::vector< double > > GMRESRP_DATA::H

(k+1 x k) upper Hessenberg storage matrix

Definition at line 200 of file lark.h.

5.41.2.6 std::vector< std::vector< double > > GMRESRP_DATA::H_bar

(k+1 x k) Factorized matrix

Definition at line 201 of file lark.h.

5.41.2.7 int GMRESRP_DATA::iter_inner = 0

Total number of inner iterations.

Definition at line 182 of file lark.h.

5.41.2.8 int GMRESRP_DATA::iter_outer = 0

Total number of outer iterations.

Definition at line 181 of file lark.h.

5.41.2.9 int GMRESRP_DATA::iter_total = 0

Total number of overall iterations.

Definition at line 183 of file lark.h.

5.41.2.10 int GMRESRP_DATA::maxit = 0

Maximum allowable outer iterations.

Definition at line 180 of file lark.h.

5.41.2.11 bool GMRESRP_DATA::Output = true

True = print messages to console.

Definition at line 192 of file lark.h.

5.41.2.12 Matrix<double> GMRESRP_DATA::r

Residual vector for the linear system.

Definition at line 196 of file lark.h.

5.41.2.13 double GMRESRP_DATA::relres

Relative residual norm for linear system.

Definition at line 188 of file lark.h.

5.41.2.14 double GMRESRP_DATA::relres_base

Initial residual norm of the linear system.

Definition at line 189 of file lark.h.

5.41.2.15 double GMRESRP_DATA::res

Absolute residual norm for linear system.

Definition at line 187 of file lark.h.

5.41.2.16 int GMRESRP_DATA::restart = -1

Restart parameter - default = min(20,vector_size)

Definition at line 179 of file lark.h.

5.41.2.17 Matrix<double> GMRESRP_DATA::sum

(N) x (1) running sum of subspace vectors for use in altering w

Definition at line 208 of file lark.h.

5.41.2.18 double GMRESRP_DATA::tol_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

Definition at line 186 of file lark.h.

5.41.2.19 double GMRESRP_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 185 of file lark.h.

5.41.2.20 Matrix<double> GMRESRP_DATA::v

(N) x (1) holding cell for the column entries of Vk and other interims

Definition at line 207 of file lark.h.

5.41.2.21 std::vector< Matrix<double> > GMRESRP_DATA::Vk

(N x k) orthonormal vector basis

Definition at line 198 of file lark.h.

5.41.2.22 Matrix<double> GMRESRP_DATA::w

(N) x (1) interim result of the matrix_vector multiplication

Definition at line 206 of file lark.h.

5.41.2.23 Matrix<double> GMRESRP_DATA::x

Current solution to the linear system.

Definition at line 194 of file lark.h.

5.41.2.24 std::vector< double > GMRESRP_DATA::y

(k x 1) Vector search direction

Definition at line 202 of file lark.h.

5.41.2.25 std::vector< Matrix<double> > GMRESRP_DATA::Zk

(N x k) preconditioned vector set

Definition at line 199 of file lark.h.

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

- lark.h

5.42 GPAST_DATA Struct Reference

GPAST Data Structure.

`#include <magpie.h>`

Public Attributes

- double x
  Adsorbed mole fraction.
- double y
  Gas phase mole fraction.
- double He
  Henry's Coefficient (mol/kg/kPa)
- double q
  Amount adsorbed for each component (mol/kg)
- std::vector< double > gama_inf
  Infinite dilution activities.
- double qo
  Pure component capacities (mol/kg)
- double Plo
  Pure component spreading pressures (mol/kg)
- std::vector< double > po
  Pure component reference state pressures (kPa)
- double poi
  Reference state pressures solved for using Recover eval GPAST.
- bool present
  If true, then the component is present; if false, then the component is not present.

5.42.1 Detailed Description

GPAST Data Structure.

C-style object holding all parameter information associated with the Generalized Predictive Adsorbed Solution Theory (GPAST) system of equations. Each species in the gas phase will have one of these objects.

Definition at line 123 of file magpie.h.

5.42.2 Member Data Documentation

5.42.2.1 std::vector<double> GPAST_DATA::gama_inf

Infinite dilution activities.

Definition at line 129 of file magpie.h.

5.42.2.2 double GPAST_DATA::He

Henry's Coefficient (mol/kg/kPa)

Definition at line 127 of file magpie.h.

5.42.2.3 double GPAST_DATA::Plo

Pure component spreading pressures (mol/kg)

Definition at line 131 of file magpie.h.

5.42.2.4 std::vector<double> GPAST_DATA::po

Pure component reference state pressures (kPa)

Definition at line 132 of file magpie.h.

5.42.2.5 double GPAST_DATA::poi

Reference state pressures solved for using Recover eval GPAST.

Definition at line 133 of file magpie.h.

5.42.2.6 bool GPAST_DATA::present

If true, then the component is present; if false, then the component is not present.

Definition at line 134 of file magpie.h.

5.42.2.7 double GPAST_DATA::q

Amount adsorbed for each component (mol/kg)

Definition at line 128 of file magpie.h.

5.42.2.8 double GPAST_DATA::qo

Pure component capacities (mol/kg)

Definition at line 130 of file magpie.h.

5.42.2.9 double GPAST_DATA::x

Adsorbed mole fraction.

Definition at line 125 of file magpie.h.

5.42.2.10 double GPAST_DATA::y

Gas phase mole fraction.

Definition at line 126 of file magpie.h.

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

- magpie.h

5.43 GSTA_DATA Struct Reference

GSTA Data Structure.

`#include <magpie.h>`

Public Attributes

- double qmax

  Theoretical maximum capacity of adsorbate-adsorbent pair (mol/kg)
- int m

  Number of parameters in the GSTA isotherm.
- std::vector< double > dHo

  Enthalpies for each site (J/mol)
- std::vector< double > dSo

  Entropies for each site (J/(K∗mol))

5.43.1 Detailed Description

GSTA Data Structure.

C-style object holding all parameter information associated with the Generalized Statistical Thermodynamic Adsorption (GSTA) isotherm model. Each species in the gas phase will have one of these objects.

Definition at line 98 of file magpie.h.

5.43.2 Member Data Documentation

5.43.2.1 std::vector<double> GSTA_DATA::dHo

Enthalpies for each site (J/mol)

Definition at line 102 of file magpie.h.

5.43.2.2 std::vector<double> GSTA_DATA::dSo

Entropies for each site (J/(K∗mol))

Definition at line 103 of file magpie.h.

5.43.2.3 int GSTA_DATA::m

Number of parameters in the GSTA isotherm.

Definition at line 101 of file magpie.h.

5.43.2.4 double GSTA_DATA::qmax

Theoretical maximum capacity of adsorbate-adsorbent pair (mol/kg)

Definition at line 100 of file magpie.h.

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

- magpie.h

5.44 KMS_DATA Struct Reference

Data structure for the implemenation of the Krylov Multi-Space (KMS) Method.

`#include <lark.h>`

Public Attributes

- int level = 0

  Current level in the recursion.
- int max_level = 0

  Maximum allowable recursion levels (Default = 0 -> GMRES, Max = 5)
- int restart = -1

  Restart parameter for the outer iterates (Default = 20, Max = N)
- int maxit = 0

  Maximum allowable iterations for the outer steps.
- int inner_iter = 0

  Number of inner steps taken.
- int outer_iter = 0

  Number of outer steps taken.
- int total_iter = 0

  Total number of iterations in all steps.
- double outer_reltol = 1e-6

  Relative residual tolerance for outer steps (Default = 1e-6)
- double outer_abstol = 1e-6

  Absolute residual tolerance for outer steps (Default = 1e-6)
- double inner_reltol = 0.1

  Residual tolerance for inner steps made relative to outer steps (Default = 0.1)
- bool Output_out = true

  True = Print the outer steps residuals.
- bool Output_in = false

  True = Print the inner steps residuals.
- GMRESRP_DATA gmres_out

  Data structure for the outer steps.
- std::vector< GMRESRP_DATA > gmres_in

  Data structures for each recursion level.
- int(* matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *matvec_data)

  User supplied matrix-vector product function.
- int(* terminal_precon)(const Matrix< double > &r, Matrix< double > &p, const void *precon_data)

  Optional user supplied terminal preconditioner.
- const void * matvec_data

  Data structure for the user's matvec function.
- const void * term_precon

  Data structure for the user's terminal preconditioner.

5.44.1 Detailed Description

Data structure for the implemenation of the Krylov Multi-Space (KMS) Method.

C-style object to be used in conjunction with the Krylov Multi-Space (KMS) Algorithm to iteratively solve non-symmetric, indefinite linear systems. This method was inspired by the Flexible GMRES (FGMRES) and Recursive GMRES (GMRESR) methods proposed by Saad (1993) and Vorst and Vuik (1991), respectively. The idea behind this method is to recursively call FGMRES to solve a linear system with pregressively smaller Krylov Subspaces built by a Right-Preconditioned GMRES algorithm. Thus creating a "V-cycle" of iteration similar to that seen in Multi-Grid algorithms.

Definition at line 413 of file lark.h.

5.44.2 Member Data Documentation

5.44.2.1 std::vector<GMRESRP_DATA> KMS_DATA::gmres_in

Data structures for each recursion level.

Definition at line 431 of file lark.h.

5.44.2.2 GMRESRP_DATA KMS_DATA::gmres_out

Data structure for the outer steps.

Definition at line 430 of file lark.h.

5.44.2.3 int KMS_DATA::inner_iter = 0

Number of inner steps taken.

Definition at line 419 of file lark.h.

5.44.2.4 double KMS_DATA::inner_reltol = 0.1

Residual tolerance for inner steps made relative to outer steps (Default = 0.1)

Definition at line 425 of file lark.h.

5.44.2.5 int KMS_DATA::level = 0

Current level in the recursion.

Definition at line 415 of file lark.h.

5.44.2.6 int(* KMS_DATA::matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *matvec_data)

User supplied matrix-vector product function.

Definition at line 434 of file lark.h.

5.44.2.7 const void* KMS_DATA::matvec_data

Data structure for the user's matvec function.

Definition at line 438 of file lark.h.

5.44.2.8 int KMS_DATA::max_level = 0

Maximum allowable recursion levels (Default = 0 -> GMRES, Max = 5)

Definition at line 416 of file lark.h.

5.44.2.9 int KMS_DATA::maxit = 0

Maximum allowable iterations for the outer steps.

Definition at line 418 of file lark.h.

5.44.2.10 double KMS_DATA::outer_abstol = 1e-6

Absolute residual tolerance for outer steps (Default = 1e-6)

Definition at line 424 of file lark.h.

5.44.2.11 int KMS_DATA::outer_iter = 0

Number of outer steps taken.

Definition at line 420 of file lark.h.

5.44.2.12 double KMS_DATA::outer_reltol = 1e-6

Relative residual tolerance for outer steps (Default = 1e-6)

Definition at line 423 of file lark.h.

5.44.2.13 bool KMS_DATA::Output_in = false

True = Print the inner steps residuals.

Definition at line 428 of file lark.h.

5.44.2.14 bool KMS_DATA::Output_out = true

True = Print the outer steps residuals.

Definition at line 427 of file lark.h.

5.44.2.15 int KMS_DATA::restart = -1

Restart parameter for the outer iterates (Default = 20, Max = N)

Definition at line 417 of file lark.h.

5.44.2.16 const void* KMS_DATA::term_precon

Data structure for the user's terminal preconditioner.

Definition at line 439 of file lark.h.

5.44.2.17 int(* KMS_DATA::terminal_precon)(const Matrix< double > &r, Matrix< double > &p, const void *precon_data)

Optional user supplied terminal preconditioner.

Definition at line 436 of file lark.h.

5.44.2.18 int KMS_DATA::total_iter = 0

Total number of iterations in all steps.

Definition at line 421 of file lark.h.

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

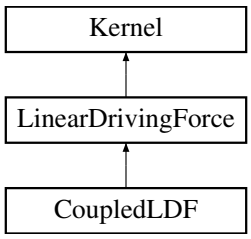
- lark.h

5.45 LinearDrivingForce Class Reference

LinearDrivingForce class object inherits from Kernel object.

`#include <LinearDrivingForce.h>`

Inheritance diagram for LinearDrivingForce:



Public Member Functions

- LinearDrivingForce (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Protected Attributes

- bool _gaining

  Boolean to mark whether the driving force is gaining or losing (True = gaining)
- Real _coef

  Coefficient for the strength or rate of the driving force.
- Real _driving_value

  Value the coupled variable is driving towards.
- VariableValue & _var

  Reference to the coupled non-linear variable.

5.45.1 Detailed Description

LinearDrivingForce class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel has several protected members including: a boolean for gaining or losing mechanisms, a coefficient for the rate or strength of the driving force, a driving value to where the coupled non-linear variable is driving toward, and the coupled non-linear variable.

Note

To create a specific linear driving force kernel, inherit from this class and use other non-linear variables or material properties to change the protected member values to reflect the physics for your problem.

Definition at line 60 of file LinearDrivingForce.h.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 LinearDrivingForce::LinearDrivingForce (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.45.3 Member Function Documentation

5.45.3.1 virtual Real LinearDrivingForce::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in CoupledLDF.

5.45.3.2 virtual Real LinearDrivingForce::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in CoupledLDF.

5.45.4 Member Data Documentation

5.45.4.1 Real LinearDrivingForce::_coef [protected]

Coefficient for the strength or rate of the driving force.

Definition at line 77 of file LinearDrivingForce.h.

5.45.4.2 Real LinearDrivingForce::_driving_value [protected]

Value the coupled variable is driving towards.

Definition at line 78 of file LinearDrivingForce.h.

5.45.4.3 bool LinearDrivingForce::_gaining [protected]

Boolean to mark whether the driving force is gaining or losing (True = gaining)

Definition at line 76 of file LinearDrivingForce.h.

5.45.4.4 VariableValue& LinearDrivingForce::_var [protected]

Reference to the coupled non-linear variable.

Definition at line 79 of file LinearDrivingForce.h.

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

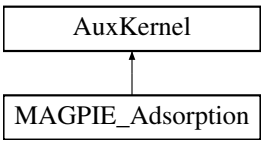
- LinearDrivingForce.h

5.46 MAGPIE_Adsorption Class Reference

Magpie Adsorption class inherits from AuxKernel.

`#include <MAGPIE_Adsorption.h>`

Inheritance diagram for MAGPIE_Adsorption:



Public Member Functions

- MAGPIE_Adsorption (const InputParameters &parameters)

  Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

Required MOOSE function override.

Private Attributes

- unsigned int _index

  Index of the gaseous species to calculate equilibria for.
- const MaterialProperty < MAGPIE_DATA > & _magpie_dat

  Material Property holding the MAGPIE data structure.

5.46.1 Detailed Description

Magpie Adsorption class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the adsorption equilibria for a given species in the gas phase based on parameters, variables, and constants set in the MAGPIE object. Those values include temperature, pressure, concentration, and associated equilibrium energy constants. The return value is the adsorption equilibrium value in mol/kg.

Definition at line 63 of file MAGPIE_Adsorption.h.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 MAGPIE_Adsorption::MAGPIE_Adsorption (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.46.3 Member Function Documentation

5.46.3.1 virtual Real MAGPIE_Adsorption::computeValue ( ) [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the AuxVariable is needed. You are required to override this function for any inherited AuxKernel.

5.46.4 Member Data Documentation

5.46.4.1 unsigned int MAGPIE_Adsorption::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 76 of file MAGPIE_Adsorption.h.

5.46.4.2 const MaterialProperty< MAGPIE_DATA >& MAGPIE_Adsorption::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 77 of file MAGPIE_Adsorption.h.

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

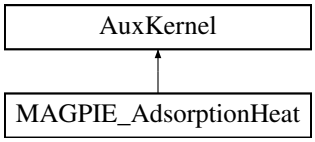
- MAGPIE_Adsorption.h

5.47 MAGPIE_AdsorptionHeat Class Reference

Magpie Adsorption Heat class inherits from AuxKernel.

`#include <MAGPIE_AdsorptionHeat.h>`

Inheritance diagram for MAGPIE_AdsorptionHeat:



Public Member Functions

- MAGPIE_AdsorptionHeat (const InputParameters &parameters)
  Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()
  Required MOOSE function override.

Private Attributes

- unsigned int _index
  Index of the gaseous species to calculate adsorbtion heat for.
- const MaterialProperty < MAGPIE_DATA > & _magpie_dat
  Material Property holding the MAGPIE data structure.
- VariableValue & _solid_conc
  Reference to the adsorbed amount of the given species (AuxVariable)

5.47.1 Detailed Description

Magpie Adsorption Heat class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the heat of adsorption for a given species in the gas phase based on parameters, variables, and constants set in the MAGPIE object. Those values include temperature, pressure, concentration, and associated equilibrium energy constants. The return value is the heat of adsorption value in J/kg.

Definition at line 63 of file MAGPIE_AdsorptionHeat.h.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 MAGPIE_AdsorptionHeat::MAGPIE_AdsorptionHeat (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.47.3 Member Function Documentation

5.47.3.1 virtual Real MAGPIE_AdsorptionHeat::computeValue ( ) [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the AuxVariable is needed. You are required to override this function for any inherited AuxKernel.

5.47.4 Member Data Documentation

5.47.4.1 unsigned int MAGPIE_AdsorptionHeat::_index [private]

Index of the gaseous species to calculate adsorbtion heat for.

Definition at line 76 of file MAGPIE_AdsorptionHeat.h.

5.47.4.2 const MaterialProperty< MAGPIE_DATA >& MAGPIE_AdsorptionHeat::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 77 of file MAGPIE_AdsorptionHeat.h.

5.47.4.3 VariableValue& MAGPIE_AdsorptionHeat::_solid_conc [private]

Reference to the adsorbed amount of the given species (AuxVariable)

Definition at line 78 of file MAGPIE_AdsorptionHeat.h.

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

- MAGPIE_AdsorptionHeat.h

5.48 MAGPIE_DATA Struct Reference

MAGPIE Data Structure.

`#include <magpie.h>`

Public Attributes

- std::vector< GSTA_DATA > gsta_dat
- std::vector< mSPD_DATA > mspd_dat
- std::vector< GPAST_DATA > gpast_dat
- SYSTEM_DATA sys_dat

5.48.1 Detailed Description

MAGPIE Data Structure.

C-style object holding all information necessary to run a MAGPIE simulation. This is the data structure that will be used in other sub-routines when a mixed gas adsorption simulation needs to be run.

Definition at line 164 of file magpie.h.

5.48.2 Member Data Documentation

5.48.2.1 std::vector<GPAST_DATA> MAGPIE_DATA::gpast_dat

Definition at line 168 of file magpie.h.

5.48.2.2 std::vector<GSTA_DATA> MAGPIE_DATA::gsta_dat

Definition at line 166 of file magpie.h.

5.48.2.3 std::vector<mSPD_DATA> MAGPIE_DATA::mspd_dat

Definition at line 167 of file magpie.h.

5.48.2.4 SYSTEM_DATA MAGPIE_DATA::sys_dat

Definition at line 169 of file magpie.h.

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

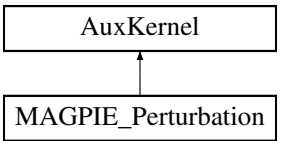
- magpie.h

5.49 MAGPIE_Perturbation Class Reference

Magpie Perturbation class inherits from AuxKernel.

`#include <MAGPIE_Perturbation.h>`

Inheritance diagram for MAGPIE_Perturbation:



Public Member Functions

- MAGPIE_Perturbation (const InputParameters &parameters)
  Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()
  Required MOOSE function override.

Private Attributes

- unsigned int _index
  Index of the gaseous species to calculate equilibria for.
- const MaterialProperty < MAGPIE_DATA > & _magpie_dat
  Material Property holding the MAGPIE data structure.

5.49.1 Detailed Description

Magpie Perturbation class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the perturbed equilibria for a given species in the gas phase based on parameters, variables, and constants set in the MAGPIE object. Those values include temperature, pressure, concentration, and associated equilibrium energy constants. The return value is the adsorption perturbation value in mol/kg.

Definition at line 72 of file MAGPIE_Perturbation.h.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 MAGPIE_Perturbation::MAGPIE_Perturbation (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.49.3 Member Function Documentation

5.49.3.1 virtual Real MAGPIE_Perturbation::computeValue ( ) [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the AuxVariable is needed. You are required to override this function for any inherited AuxKernel.

5.49.4 Member Data Documentation

5.49.4.1 unsigned int MAGPIE_Perturbation::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 85 of file MAGPIE_Perturbation.h.

5.49.4.2 const MaterialProperty< MAGPIE_DATA >& MAGPIE_Perturbation::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 86 of file MAGPIE_Perturbation.h.

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

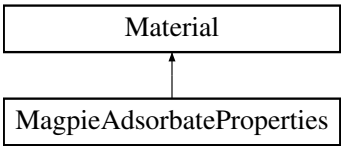
- MAGPIE_Perturbation.h

5.50 MagpieAdsorbateProperties Class Reference

MagpieAdsorbateProperties class object inherits from Material object.

`#include <MagpieAdsorbateProperties.h>`

Inheritance diagram for MagpieAdsorbateProperties:



Public Member Functions

- MagpieAdsorbateProperties (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual void computeQpProperties ()

  Required function override for Material objects in MOOSE.

Private Attributes

- std::vector< unsigned int > _index

  Indices for the gas species in the system.
- VariableValue & _temperature

Reference to the coupled column temperature.

- VariableValue & _total_pressure

  Reference to the coupled column pressure.

- std::vector< VariableValue * > _gas_conc

  Pointer list to the coupled gases.

- std::vector< VariableValue * > _gas_conc_old

  Pointer list to the old states of coupled gases.

- std::vector< int > _num_sites

  List of the number of sites each gas species' isotherm contains.

- std::vector< Real > _max_capacity

  List of the maximum adsorption capacities of each gas species.

- std::vector< Real > _molar_volume

  List of the van der Waal's molar volumes of each species.

- std::vector< Real > _enthalpy_1

  List of the site 1 enthalpies for each gas species.

- std::vector< Real > _enthalpy_2

  List of the site 2 enthalpies for each gas species.

- std::vector< Real > _enthalpy_3

  List of the site 3 enthalpies for each gas species.

- std::vector< Real > _enthalpy_4

  List of the site 4 enthalpies for each gas species.

- std::vector< Real > _enthalpy_5

  List of the site 5 enthalpies for each gas species.

- std::vector< Real > _enthalpy_6

  List of the site 6 enthalpies for each gas species.

- std::vector< Real > _entropy_1

  List of the site 1 entropies for each gas species.

- std::vector< Real > _entropy_2

  List of the site 2 entropies for each gas species.

- std::vector< Real > _entropy_3

  List of the site 3 entropies for each gas species.

- std::vector< Real > _entropy_4

  List of the site 4 entropies for each gas species.

- std::vector< Real > _entropy_5

  List of the site 5 entropies for each gas species.

- std::vector< Real > _entropy_6

  List of the site 6 entropies for each gas species.

- MaterialProperty< MAGPIE_DATA > & _magpie_dat

  MaterialProperty object to hold the MAGPIE_DATA structure and all relavent information.

5.50.1 Detailed Description

MagpieAdsorbateProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up the MAGPIE_DATA structure (see magpie.h) based on user provied input from the input file. That information will be used to estimate the adsorption of each species in the system based on temperature, pressure, and concentrations of each species.

Note

The GSTA isotherm model for each species allows upto 6 energetically distinct adsorption sites. If those sites are not used by a particular species, then those energies should be left as zeros in the input files and the number of relavent sites for each species needs to be recorded in the input file. Each species is allowed to have a different number of adsorption sites in a particular adsorbent.

Definition at line 62 of file MagpieAdsorbateProperties.h.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 MagpieAdsorbateProperties::MagpieAdsorbateProperties (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.50.3 Member Function Documentation

5.50.3.1 virtual void MagpieAdsorbateProperties::computeQpProperties ( ) [protected],[virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

5.50.4 Member Data Documentation

5.50.4.1 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_1 [private]

List of the site 1 enthalpies for each gas species.

Definition at line 85 of file MagpieAdsorbateProperties.h.

5.50.4.2 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_2 [private]

List of the site 2 enthalpies for each gas species.

Definition at line 86 of file MagpieAdsorbateProperties.h.

5.50.4.3 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_3 [private]

List of the site 3 enthalpies for each gas species.

Definition at line 87 of file MagpieAdsorbateProperties.h.

5.50.4.4 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_4 [private]

List of the site 4 enthalpies for each gas species.

Definition at line 88 of file MagpieAdsorbateProperties.h.

5.50.4.5 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_5 [private]

List of the site 5 enthalpies for each gas species.

Definition at line 89 of file MagpieAdsorbateProperties.h.

5.50.4.6 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_6 [private]

List of the site 6 enthalpies for each gas species.

Definition at line 90 of file MagpieAdsorbateProperties.h.

5.50.4.7 std::vector<Real> MagpieAdsorbateProperties::_entropy_1 [private]

List of the site 1 entropies for each gas species.

Definition at line 92 of file MagpieAdsorbateProperties.h.

5.50.4.8 std::vector<Real> MagpieAdsorbateProperties::_entropy_2 [private]

List of the site 2 entropies for each gas species.

Definition at line 93 of file MagpieAdsorbateProperties.h.

5.50.4.9 std::vector<Real> MagpieAdsorbateProperties::_entropy_3 [private]

List of the site 3 entropies for each gas species.

Definition at line 94 of file MagpieAdsorbateProperties.h.

5.50.4.10 std::vector<Real> MagpieAdsorbateProperties::_entropy_4 [private]

List of the site 4 entropies for each gas species.

Definition at line 95 of file MagpieAdsorbateProperties.h.

5.50.4.11 std::vector<Real> MagpieAdsorbateProperties::_entropy_5 [private]

List of the site 5 entropies for each gas species.

Definition at line 96 of file MagpieAdsorbateProperties.h.

5.50.4.12 std::vector<Real> MagpieAdsorbateProperties::_entropy_6 [private]

List of the site 6 entropies for each gas species.

Definition at line 97 of file MagpieAdsorbateProperties.h.

5.50.4.13 std::vector<VariableValue *> MagpieAdsorbateProperties::_gas_conc [private]

Pointer list to the coupled gases.

Definition at line 78 of file MagpieAdsorbateProperties.h.

5.50.4.14 std::vector<VariableValue *> MagpieAdsorbateProperties::_gas_conc_old [private]

Pointer list to the old states of coupled gases.

Definition at line 79 of file MagpieAdsorbateProperties.h.

5.50.4.15 std::vector<unsigned int> MagpieAdsorbateProperties::_index [private]

Indices for the gas species in the system.

Definition at line 75 of file MagpieAdsorbateProperties.h.

5.50.4.16 MaterialProperty< MAGPIE_DATA >& MagpieAdsorbateProperties::_magpie_dat [private]

MaterialProperty object to hold the MAGPIE_DATA structure and all relavent information.

This is the object that needs to interface with the MAGPIE functions in order to solve for variable information such as adsorption capacities, mixed gas adsorption equilibria, and heats of adsorption.

Definition at line 103 of file MagpieAdsorbateProperties.h.

5.50.4.17 std::vector<Real> MagpieAdsorbateProperties::_max_capacity [private]

List of the maximum adsorption capacities of each gas species.

Definition at line 82 of file MagpieAdsorbateProperties.h.

5.50.4.18 std::vector<Real> MagpieAdsorbateProperties::_molar_volume [private]

List of the van der Waal's molar volumes of each species.

Definition at line 83 of file MagpieAdsorbateProperties.h.

5.50.4.19 std::vector<int> MagpieAdsorbateProperties::_num_sites [private]

List of the number of sites each gas species' isotherm contains.

Definition at line 81 of file MagpieAdsorbateProperties.h.

5.50.4.20 VariableValue& MagpieAdsorbateProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 76 of file MagpieAdsorbateProperties.h.

5.50.4.21 VariableValue& MagpieAdsorbateProperties::_total_pressure [private]

Reference to the coupled column pressure.

Definition at line 77 of file MagpieAdsorbateProperties.h.

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

- MagpieAdsorbateProperties.h

5.51 Matrix< T > Class Template Reference

Templated C++ Matrix Class Object (click Matrix to go to function definitions)

`#include <macaw.h>`

Public Member Functions

- Matrix (int rows, int columns)
  Constructor for matrix with given number of rows and columns.
- T & operator() (int i, int j)
  Access operator for the matrix element at row i and column j (e.g., aij = A(i,j))
- T operator() (int i, int j) const
  Constant access operator for the the matrix element at row i and column j.
- Matrix (const Matrix &M)
  Copy constructor for constructing a matrix as a copy of another matrix.
- Matrix & operator= (const Matrix &M)
  Equals operator for setting one matrix equal to another matrix.
- Matrix ()
  Default constructor for creating an empty matrix.
- ~Matrix ()
  Default destructor for clearing out memory.
- void set_size (int i, int j)
  Function to set/change the size of a matrix to i rows and j columns.
- void zeros ()
  Function to set/change all values in a matrix to zeros.
- void edit (int i, int j, T value)
  Function to set/change the element of a matrix at row i and column j to given value.

- int rows ()

  Function to return the number of rows in a given matrix.
- int columns ()

  Function to return the number of columns in a matrix.
- T determinate ()

  Function to compute the determinate of a matrix and return that value.
- T norm ()

  Function to compute the L2-norm of a matrix and return that value.
- T sum ()

  Function to compute the sum of all elements in a matrix and return that value.
- T inner_product (const Matrix &x)

  Function to compute the inner product between this matrix and matrix x.
- Matrix & cofactor (const Matrix &M)

  Function to convert this matrix to a cofactor matrix of the given matrix M.
- Matrix operator+ (const Matrix &M)

  Operator to add this matrix and matrix M and return the new matrix result.
- Matrix operator- (const Matrix &M)

  Operator to subtract this matrix and matrix M and return the new matrix result.
- Matrix operator∗ (const T)

  Operator to multiply this matrix by a scalar T return the new matrix result.
- Matrix operator/ (const T)

  Operator to divide this matrix by a scalar T and return the new matrix result.
- Matrix operator∗ (const Matrix &M)

  Operator to multiply this matrix and matrix M and return the new matrix result.
- Matrix & transpose (const Matrix &M)

  Function to convert this matrix to the transpose of the given matrix M.
- Matrix & transpose_multiply (const Matrix &MT, const Matrix &v)

  Function to convert this matrix into the result of the given matrix M transposed and multiplied by the other given matrix v.
- Matrix & adjoint (const Matrix &M)

  Function to convert this matrix to the adjoint of the given matrix.
- Matrix & inverse (const Matrix &M)

  Function to convert this matrix to the inverse of the given matrix.
- void Display (const std::string Name)

  Function to display the contents of this matrix given a Name for the matrix.
- Matrix & tridiagonalSolve (const Matrix &A, const Matrix &b)

  Function to solve Ax=b for x if A is symmetric, tridiagonal (this->x)
- Matrix & ladshawSolve (const Matrix &A, const Matrix &d)

  Function to solve Ax=d for x if A is non-symmetric, tridiagonal (this->x)
- Matrix & tridiagonalFill (const T A, const T B, const T C, bool Spherical)

  Function to fill in this matrix with coefficients A, B, and C to form a tridiagonal matrix.
- Matrix & naturalLaplacian3D (int m)

  Function to fill out this matrix with coefficients from a 3D Laplacian function.
- Matrix & sphericalBCFill (int node, const T coeff, T variable)

  Function to fill out a column matrix with spherical specific boundary conditions.
- Matrix & ConstantICFill (const T IC)

  Function to set all values in a column matrix to a given constant.
- Matrix & SolnTransform (const Matrix &A, bool Forward)

  Function to transform the values in a column matrix from cartesian to spherical coordinates.
- T sphericalAvg (double radius, double dr, double bound, bool Dirichlet)

  Function to compute a spatial average of this column matrix in spherical coordinates.

- T IntegralAvg (double radius, double dr, double bound, bool Dirichlet)

  Function to compute a spatial average of this column matrix in spherical coordinates.
- T IntegralTotal (double dr, double bound, bool Dirichlet)

  Function to compute a spatial total of this column matrix in spherical coordinates.
- Matrix & tridiagonalVectorFill (const std::vector< T > &A, const std::vector< T > &B, const std::vector< T > &C)

  Function to fill in this matrix, in tridiagonal fashion, using the vectors of coefficients.
- Matrix & columnVectorFill (const std::vector< T > &A)

  Function to fill in a column matrix with the values of the given vector object.
- Matrix & columnProjection (const Matrix &b, const Matrix &b_old, const double dt, const double dt_old)

  Function to project a column matrix solution in time based on older state vectors.
- Matrix & dirichletBCFill (int node, const T coeff, T variable)

  Function to fill in a column matrix with all zeros except at the given node.
- Matrix & diagonalSolve (const Matrix &D, const Matrix &v)

  Function to solve the system Dx=v for x given that D is diagonal (this->x)
- Matrix & upperTriangularSolve (const Matrix &U, const Matrix &v)

  Function to solve the system Ux=v for x given that U is upper Triagular (this->x)
- Matrix & lowerTriangularSolve (const Matrix &L, const Matrix &v)

  Function to solve the system Lx=v for x given that L is lower Triagular (this->x)
- Matrix & upperHessenberg2Triangular (Matrix &b)

  Function to convert this square matrix to upper Triangular (assuming this is upper Hessenberg)
- Matrix & lowerHessenberg2Triangular (Matrix &b)

  Function to convert this square matrix to lower Triangular (assuming this is lower Hessenberg)
- Matrix & upperHessenbergSolve (const Matrix &H, const Matrix &v)

  Function to solve the system Hx=v for x given that H is upper Hessenberg (this->x)
- Matrix & lowerHessenbergSolve (const Matrix &H, const Matrix &v)

  Function to solve the system Hx=v for x given that H is lower Hessenberg (this->x)
- Matrix & columnExtract (int j, const Matrix &M)

  Function to set this column matrix to the jth column of the given matrix M.
- Matrix & rowExtract (int i, const Matrix &M)

  Function to set this row matrix to the ith row of the given matrix M.
- Matrix & columnReplace (int j, const Matrix &v)

  Function to this matrices' jth column with the given column matrix v.
- Matrix & rowReplace (int i, const Matrix &v)

  Function to this matrices' ith row with the given row matrix v.
- void rowShrink ()

  Function to delete the last row of this matrix.
- void columnShrink ()

  Function to delete the last column of this matrix.
- void rowExtend (const Matrix &v)

  Function to add the row matrix v to the end of this matrix.
- void columnExtend (const Matrix &v)

  Function to add the column matrix v to the end of this matrix.

Protected Attributes

- int num_rows

  Number of rows of the matrix.
- int num_cols

  Number of columns of the matrix.
- std::vector< T > Data

  Storage vector for the elements of the matrix.

5.51.1 Detailed Description

template<class T>class Matrix< T >

Templated C++ Matrix Class Object (click Matrix to go to function definitions)

C++ templated class object containing many different functions, actions, and solver routines associated with Dense Matrices. Operator overloads are also provided to give the user a more natural way of operating matrices on other matrices or scalars. These operator overloads are especially useful for reducing the amount of code needed to be written when working with matrix-based problems.

Definition at line 53 of file macaw.h.

5.51.2 Constructor & Destructor Documentation

5.51.2.1 template<class T > Matrix< T >::Matrix (int *rows*, int *columns*)

Constructor for matrix with given number of rows and columns.

Definition at line 208 of file macaw.h.

5.51.2.2 template<class T > Matrix< T >::Matrix (const Matrix< T > & *M*)

Copy constructor for constructing a matrix as a copy of another matrix.

Definition at line 247 of file macaw.h.

References Matrix< T >::Data, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.2.3 template<class T > Matrix< T >::Matrix ( )

Default constructor for creating an empty matrix.

Definition at line 292 of file macaw.h.

References Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.2.4 template<class T > Matrix< T >::~Matrix ( )

Default destructor for clearing out memory.

Definition at line 302 of file macaw.h.

5.51.3 Member Function Documentation

5.51.3.1 template<class T > Matrix< T > & Matrix< T >::adjoint (const Matrix< T > & *M*)

Function to convert this matrix to the adjoint of the given matrix.

Definition at line 734 of file macaw.h.

References arg_matrix_same, mError, non_square_matrix, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.2 template<class T > Matrix< T > & Matrix< T >::cofactor (const Matrix< T > & *M*)

Function to convert this matrix to a cofactor matrix of the given matrix M.

Definition at line 489 of file macaw.h.

References arg_matrix_same, Matrix< T >::Data, Matrix< T >::determinate(), mError, non_square_matrix, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.3 template<class T > void Matrix< T >::columnExtend (const Matrix< T > & *v*)

Function to add the column matrix v to the end of this matrix.

Definition at line 1774 of file macaw.h.

References matvec_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.4 template<class T > Matrix< T > & Matrix< T >::columnExtract (int *j,* const Matrix< T > & *M*)

Function to set this column matrix to the jth column of the given matrix M.

Definition at line 1644 of file macaw.h.

References arg_matrix_same, mError, and Matrix< T >::num_rows.

5.51.3.5 template<class T > Matrix< T > & Matrix< T >::columnProjection (const Matrix< T > & *b,* const Matrix< T > & *b_old,* const double *dt,* const double *dt_old*)

Function to project a column matrix solution in time based on older state vectors.

This function is used in finch.h to form Matrix u_star. It uses the size of the current step and old step, dt and dt_old respectively, to form an approximation for the next state. The current state and olde state of the variables are passed as b and b_old respectively.

Definition at line 1344 of file macaw.h.

References arg_matrix_same, dim_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.6 template<class T > Matrix< T > & Matrix< T >::columnReplace (int *j,* const Matrix< T > & *v*)

Function to this matrices' jth column with the given column matrix v.

Definition at line 1686 of file macaw.h.

References arg_matrix_same, matvec_mis_match, mError, and Matrix< T >::num_rows.

5.51.3.7 template<class T > int Matrix< T >::columns ( )

Function to return the number of columns in a matrix.

Definition at line 351 of file macaw.h.

5.51.3.8 template<class T > void Matrix< T >::columnShrink ( )

Function to delete the last column of this matrix.

Definition at line 1741 of file macaw.h.

References Matrix< T >::Data, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.9 template<class T> Matrix< T > & Matrix< T >::columnVectorFill (const std::vector< T > & *A*)

Function to fill in a column matrix with the values of the given vector object.

Definition at line 1322 of file macaw.h.

References dim_mis_match, matvec_mis_match, and mError.

5.51.3.10 template<class T> Matrix< T > & Matrix< T >::ConstantICFill (const T *IC*)

Function to set all values in a column matrix to a given constant.

Definition at line 1134 of file macaw.h.

References dim_mis_match, and mError.

5.51.3.11 template<class T > T Matrix< T >::determinate ( )

Function to compute the determinate of a matrix and return that value.

Definition at line 358 of file macaw.h.

References Matrix< T >::Data, Matrix< T >::determinate(), mError, non_square_matrix, Matrix< T >::num_cols, and Matrix< T >::num_rows.

Referenced by Matrix< T >::cofactor(), Matrix< T >::determinate(), and Matrix< T >::inverse().

5.51.3.12 template<class T > Matrix< T > & Matrix< T >::diagonalSolve (const Matrix< T > & *D,* const Matrix< T > & *v*)

Function to solve the system Dx=v for x given that D is diagonal (this->x)

Definition at line 1395 of file macaw.h.

References arg_matrix_same, dim_mis_match, mError, Matrix< T >::num_cols, Matrix< T >::num_rows, and singular_matrix.

5.51.3.13 template<class T> Matrix< T > & Matrix< T >::dirichletBCFill (int *node,* const T *coeff,* T *variable*)

Function to fill in a column matrix with all zeros except at the given node.

Similar to sphericalBCFill, this function will set the values of all elements in the column matrix to zero except at the given node, where the value is set to the product of coeff and variable. This is often used to set BCs in finch.h or other related files/simulations.

Definition at line 1369 of file macaw.h.

References dim_mis_match, and mError.

5.51.3.14 template<class T > void Matrix< T >::Display (const std::string *Name*)

Function to display the contents of this matrix given a Name for the matrix.

Definition at line 782 of file macaw.h.

References empty_matrix, and mError.

5.51.3.15 template<class T> void Matrix< T >::edit (int *i,* int *j,* T *value*)

Function to set/change the element of a matrix at row i and column j to given value.

Definition at line 332 of file macaw.h.

References mError, Matrix< T >::operator()(), and out_of_bounds.

Referenced by Matrix< T >::lowerHessenberg2Triangular(), Matrix< T >::operator*(), and Matrix< T >::upperHessenberg2Triangular().

5.51.3.16 template<class T > T Matrix< T >::inner_product (const Matrix< T > & *x*)

Function to compute the inner product between this matrix and matrix x.

Definition at line 463 of file macaw.h.

References dim_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.17 template<class T > T Matrix< T >::IntegralAvg (double *radius,* double *dr,* double *bound,* bool *Dirichlet*)

Function to compute a spatial average of this column matrix in spherical coordinates.

This function is used to compute an average value of a variable, represented in this column matrix, by integrating over the domain of the sphere. (Assumes you DO NOT have variable value at center node)

Parameters

radius	radius of the sphere
dr	space between each node
bound	value of the variable at the boundary
Dirichlet	True if problem has a Dirichlet BC, False if Neumann

Definition at line 1182 of file macaw.h.

References dim_mis_match, mError, and qo().

5.51.3.18 template<class T > T Matrix< T >::IntegralTotal (double *dr,* double *bound,* bool *Dirichlet*)

Function to compute a spatial total of this column matrix in spherical coordinates.

This function is used to compute an average value of a variable, represented in this column matrix, by integrating over the domain of the sphere. (Assumes you DO NOT have variable value at center node)

Parameters

dr	space between each node
bound	value of the variable at the boundary
Dirichlet	True if problem has a Dirichlet BC, False if Neumann

Definition at line 1242 of file macaw.h.

References dim_mis_match, M_PI, mError, and qo().

5.51.3.19 template<class T > Matrix< T > & Matrix< T >::inverse (const Matrix< T > & *M*)

Function to convert this matrix to the inverse of the given matrix.

Definition at line 756 of file macaw.h.

References A, arg_matrix_same, Matrix< T >::Data, Matrix< T >::determinate(), mError, non_square_matrix, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.20 template<class T > Matrix< T > & Matrix< T >::ladshawSolve (const Matrix< T > & *A,* const Matrix< T > & *d*)

Function to solve Ax=d for x if A is non-symmetric, tridiagonal (this->x)

Definition at line 876 of file macaw.h.

References A, arg_matrix_same, dim_mis_match, mError, Matrix< T >::num_cols, Matrix< T >::num_rows, singular_matrix, and unstable_matrix.

5.51.3.21 template<class T > Matrix< T > & Matrix< T >::lowerHessenberg2Triangular (Matrix< T > & *b*)

Function to convert this square matrix to lower Triangular (assuming this is lower Hessenberg)

During this transformation, a column vector (b) is also being transformed to represent the BCs in a linear system. This algorithm uses Givens Rotations to efficiently convert the lower Hessenberg matrix to an lower triangular matrix.

Definition at line 1561 of file macaw.h.

References arg_matrix_same, dim_mis_match, Matrix< T >::edit(), matrix_too_small, mError, Matrix< T >::num_cols, Matrix< T >::num_rows, and singular_matrix.

5.51.3.22 template<class T > Matrix< T > & Matrix< T >::lowerHessenbergSolve (const Matrix< T > & *H,* const Matrix< T > & *v*)

Function to solve the system Hx=v for x given that H is lower Hessenberg (this->x)

Definition at line 1627 of file macaw.h.

References arg_matrix_same, Matrix< T >::Data, and mError.

5.51.3.23 template<class T > Matrix< T > & Matrix< T >::lowerTriangularSolve (const Matrix< T > & *L,* const Matrix< T > & *v*)

Function to solve the system Lx=v for x given that L is lower Triagular (this->x)

Definition at line 1471 of file macaw.h.

References arg_matrix_same, dim_mis_match, mError, Matrix< T >::num_cols, Matrix< T >::num_rows, and singular_matrix.

5.51.3.24 template<class T > Matrix< T > & Matrix< T >::naturalLaplacian3D (int *m*)

Function to fill out this matrix with coefficients from a 3D Laplacian function.

This function will fill out the coefficients of the matrix with the coefficients that stem from discretizing a 3D Laplacian on a natural grid with 2nd order finite differences.

Definition at line 1031 of file macaw.h.

5.51.3.25 template<class T > T Matrix< T >::norm ( )

Function to compute the L2-norm of a matrix and return that value.

Definition at line 427 of file macaw.h.

5.51.3.26 template<class T > T & Matrix< T >::operator() (int *i*, int *j*)

Access operator for the matrix element at row i and column j (e.g., aij = A(i,j))

Definition at line 219 of file macaw.h.

References mError, and out_of_bounds.

Referenced by Matrix< T >::edit().

5.51.3.27 template<class T > T Matrix< T >::operator() (int *i*, int *j*) const

Constant access operator for the the matrix element at row i and column j.

Definition at line 232 of file macaw.h.

References mError, and out_of_bounds.

5.51.3.28 template<class T> Matrix< T > Matrix< T >::operator* (const T *a*)

Operator to multiply this matrix by a scalar T return the new matrix result.

Definition at line 612 of file macaw.h.

References Matrix< T >::Data.

5.51.3.29 template<class T> Matrix< T > Matrix< T >::operator* (const Matrix< T > & *M*)

Operator to multiply this matrix and matrix M and return the new matrix result.

Definition at line 642 of file macaw.h.

References Matrix< T >::Data, dim_mis_match, Matrix< T >::edit(), mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.30 template<class T > Matrix< T > Matrix< T >::operator+ (const Matrix< T > & *M*)

Operator to add this matrix and matrix M and return the new matrix result.

Definition at line 566 of file macaw.h.

References Matrix< T >::Data, dim_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.31 template<class T > Matrix< T > Matrix< T >::operator- (const Matrix< T > & *M*)

Operator to subtract this matrix and matrix M and return the new matrix result.

Definition at line 589 of file macaw.h.

References Matrix< T >::Data, dim_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.32 template<class T> Matrix< T > Matrix< T >::operator/ (const T *a*)

Operator to divide this matrix by a scalar T and return the new matrix result.

Definition at line 627 of file macaw.h.

References Matrix< T >::Data.

5.51.3.33 template<class T > Matrix< T > & Matrix< T >::operator= (const Matrix< T > & *M*)

Equals operator for setting one matrix equal to another matrix.

Definition at line 264 of file macaw.h.

References Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.34 template<class T > void Matrix< T >::rowExtend (const Matrix< T > & *v*)

Function to add the row matrix v to the end of this matrix.

Definition at line 1758 of file macaw.h.

References matvec_mis_match, mError, and Matrix< T >::num_cols.

5.51.3.35 template<class T > Matrix< T > & Matrix< T >::rowExtract (int *i,* const Matrix< T > & *M*)

Function to set this row matrix to the ith row of the given matrix M.

Definition at line 1665 of file macaw.h.

References arg_matrix_same, mError, and Matrix< T >::num_cols.

5.51.3.36 template<class T > Matrix< T > & Matrix< T >::rowReplace (int *i,* const Matrix< T > & *v*)

Function to this matrices' ith row with the given row matrix v.

Definition at line 1707 of file macaw.h.

References arg_matrix_same, matvec_mis_match, mError, and Matrix< T >::num_cols.

5.51.3.37 template<class T > int Matrix< T >::rows ( )

Function to return the number of rows in a given matrix.

Definition at line 344 of file macaw.h.

5.51.3.38 template<class T > void Matrix< T >::rowShrink ( )

Function to delete the last row of this matrix.

Definition at line 1728 of file macaw.h.

5.51.3.39 template<class T > void Matrix< T >::set_size (int *i,* int *j*)

Function to set/change the size of a matrix to i rows and j columns.

Definition at line 309 of file macaw.h.

References invalid_size, and mError.

5.51.3.40 template<class T > Matrix< T > & Matrix< T >::SolnTransform (const Matrix< T > & *A,* bool *Forward*)

Function to transform the values in a column matrix from cartesian to spherical coordinates.

Definition at line 1153 of file macaw.h.

References arg_matrix_same, Matrix< T >::Data, dim_mis_match, mError, and Matrix< T >::num_rows.

5.51.3.41 template<class T > T Matrix< T >::sphericalAvg (double *radius,* double *dr,* double *bound,* bool *Dirichlet*)

Function to compute a spatial average of this column matrix in spherical coordinates.

This function is used to compute an average value of a variable, represented in this column matrix, by integrating over the domain of the sphere. (Assumes you have variable value at center node)

Parameters

radius	radius of the sphere
dr	space between each node
bound	value of the variable at the boundary
Dirichlet	True if problem has a Dirichlet BC, False if Neumann

Definition at line 1220 of file macaw.h.

References dim_mis_match, and mError.

5.51.3.42 template<class T> Matrix< T > & Matrix< T >::sphericalBCFill (int *node,* const T *coeff,* T *variable*)

Function to fill out a column matrix with spherical specific boundary conditions.

This function will fille out a column matrix with zeros at all nodes expect for the node indicated. That node's value will be the product of the node id with the coeff and variable values given.

Definition at line 1108 of file macaw.h.

References dim_mis_match, and mError.

5.51.3.43 template<class T > T Matrix< T >::sum ( )

Function to compute the sum of all elements in a matrix and return that value.

Definition at line 448 of file macaw.h.

5.51.3.44 template<class T > Matrix< T > & Matrix< T >::transpose (const Matrix< T > & *M*)

Function to convert this matrix to the transpose of the given matrix M.

Definition at line 676 of file macaw.h.

References arg_matrix_same, Matrix< T >::Data, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.45 template<class T > Matrix< T > & Matrix< T >::transpose_multiply (const Matrix< T > & *MT,* const Matrix< T > & *v*)

Function to convert this matrix into the result of the given matrix M transposed and multiplied by the other given matrix v.

Definition at line 699 of file macaw.h.

References arg_matrix_same, Matrix< T >::Data, dim_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.46 template<class T> Matrix< T > & Matrix< T >::tridiagonalFill (const T *A,* const T *B,* const T *C,* bool *Spherical*)

Function to fill in this matrix with coefficients A, B, and C to form a tridiagonal matrix.

This function fills in the diagonal elements of a square matrix with coefficient B, upper diagonal with C, and lower diagonal with A. The boolean will apply a transformation to those coefficients, if the problem happens to stem from 1-D diffusion in spherical coordinates.

Definition at line 981 of file macaw.h.

References mError, and non_square_matrix.

5.51.3.47 template<class T > Matrix< T > & Matrix< T >::tridiagonalSolve (const Matrix< T > & *A,* const Matrix< T > & *b*)

Function to solve Ax=b for x if A is symmetric, tridiagonal (this->x)

Definition at line 807 of file macaw.h.

References A, arg_matrix_same, dim_mis_match, mError, Matrix< T >::num_cols, and Matrix< T >::num_rows.

5.51.3.48 template<class T> Matrix< T > & Matrix< T >::tridiagonalVectorFill (const std::vector< T > & *A,* const std::vector< T > & *B,* const std::vector< T > & *C*)

Function to fill in this matrix, in tridiagonal fashion, using the vectors of coefficients.

Definition at line 1279 of file macaw.h.

References matvec_mis_match, mError, and non_square_matrix.

5.51.3.49 template<class T > Matrix< T > & Matrix< T >::upperHessenberg2Triangular (Matrix< T > & *b*)

Function to convert this square matrix to upper Triangular (assuming this is upper Hessenberg)

During this transformation, a column vector (b) is also being transformed to represent the BCs in a linear system. This algorithm uses Givens Rotations to efficiently convert the upper Hessenberg matrix to an upper triangular matrix.

Definition at line 1512 of file macaw.h.

References arg_matrix_same, dim_mis_match, Matrix< T >::edit(), matrix_too_small, mError, Matrix< T >::num_cols, Matrix< T >::num_rows, and singular_matrix.

5.51.3.50 template<class T > Matrix< T > & Matrix< T >::upperHessenbergSolve (const Matrix< T > & *H,* const Matrix< T > & *v*)

Function to solve the system Hx=v for x given that H is upper Hessenberg (this->x)

Definition at line 1610 of file macaw.h.

References arg_matrix_same, Matrix< T >::Data, and mError.

5.51.3.51 template<class T > Matrix< T > & Matrix< T >::upperTriangularSolve (const Matrix< T > & *U,* const Matrix< T > & *v*)

Function to solve the system Ux=v for x given that U is upper Triagular (this->x)

Definition at line 1430 of file macaw.h.

References arg_matrix_same, dim_mis_match, mError, Matrix< T >::num_cols, Matrix< T >::num_rows, and singular_matrix.

5.51.3.52 template<class T > void Matrix< T >::zeros ( )

Function to set/change all values in a matrix to zeros.

Definition at line 324 of file macaw.h.

5.51.4 Member Data Documentation

5.51.4.1 template<class T> std::vector<T> Matrix< T >::Data [protected]

Storage vector for the elements of the matrix.

Definition at line 203 of file macaw.h.

Referenced by Matrix< T >::cofactor(), Matrix< T >::columnShrink(), Matrix< T >::determinate(), Matrix< T >::inverse(), Matrix< T >::lowerHessenbergSolve(), Matrix< T >::Matrix(), Matrix< T >::operator*(), Matrix< T

>::operator+(), Matrix< T >::operator-(), Matrix< T >::operator/(), Matrix< T >::SolnTransform(), Matrix< T >::transpose(), Matrix< T >::transpose_multiply(), and Matrix< T >::upperHessenbergSolve().

5.51.4.2 template<class T> int Matrix< T >::num_cols [protected]

Number of columns of the matrix.

Definition at line 202 of file macaw.h.

Referenced by Matrix< T >::adjoint(), Matrix< T >::cofactor(), Matrix< T >::columnExtend(), Matrix< T >::columnProjection(), Matrix< T >::columnShrink(), Matrix< T >::determinate(), Matrix< T >::diagonalSolve(), Matrix< T >::inner_product(), Matrix< T >::inverse(), Matrix< T >::ladshawSolve(), Matrix< T >::lowerHessenberg2Triangular(), Matrix< T >::lowerTriangularSolve(), Matrix< T >::Matrix(), Matrix< T >::operator*(), Matrix< T >::operator+(), Matrix< T >::operator-(), Matrix< T >::operator=(), Matrix< T >::rowExtend(), Matrix< T >::rowExtract(), Matrix< T >::rowReplace(), Matrix< T >::transpose(), Matrix< T >::transpose_multiply(), Matrix< T >::tridiagonalSolve(), Matrix< T >::upperHessenberg2Triangular(), and Matrix< T >::upperTriangularSolve().

5.51.4.3 template<class T> int Matrix< T >::num_rows [protected]

Number of rows of the matrix.

Definition at line 201 of file macaw.h.

Referenced by Matrix< T >::adjoint(), Matrix< T >::cofactor(), Matrix< T >::columnExtend(), Matrix< T >::columnExtract(), Matrix< T >::columnProjection(), Matrix< T >::columnReplace(), Matrix< T >::columnShrink(), Matrix< T >::determinate(), Matrix< T >::diagonalSolve(), Matrix< T >::inner_product(), Matrix< T >::inverse(), Matrix< T >::ladshawSolve(), Matrix< T >::lowerHessenberg2Triangular(), Matrix< T >::lowerTriangularSolve(), Matrix< T >::Matrix(), Matrix< T >::operator*(), Matrix< T >::operator+(), Matrix< T >::operator-(), Matrix< T >::operator=(), Matrix< T >::SolnTransform(), Matrix< T >::transpose(), Matrix< T >::transpose_multiply(), Matrix< T >::tridiagonalSolve(), Matrix< T >::upperHessenberg2Triangular(), and Matrix< T >::upperTriangularSolve().

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

- macaw.h

5.52 MIXED_GAS Struct Reference

Data structure holding information necessary for computing mixed gas properties.

`#include <egret.h>`

Public Attributes

- int N

  Given: Total number of gas species.
- bool CheckMolefractions = true

  Given: True = Check Molefractions for errors.
- double total_pressure

  Given: Total gas pressure (kPa)
- double gas_temperature

  Given: Gas temperature (K)
- double velocity

  Given: Gas phase velocity (cm/s)
- double char_length

  Given: Characteristic Length (cm)
- std::vector< double > molefraction

  Given: Gas molefractions of each species (-)

- double total_density

  Calculated: Total gas density (g/cm^3) {use RE3}.
- double total_dyn_vis

  Calculated: Total dynamic viscosity (g/cm/s)
- double kinematic_viscosity

  Calculated: Kinematic viscosity (cm^2/s)
- double total_molecular_weight

  Calculated: Total molecular weight (g/mol)
- double total_specific_heat

  Calculated: Total specific heat (J/g/K)
- double Reynolds

  Calculated: Value of the Reynold's number (-)
- Matrix< double > binary_diffusion

  Calculated: Tensor matrix of binary gas diffusivities (cm^2/s)
- std::vector< PURE_GAS > species_dat

  Vector of the pure gas info of all species.

5.52.1 Detailed Description

Data structure holding information necessary for computing mixed gas properties.

C-style object holding the mixed gas information necessary for performing gas dynamic simulations. This object works in conjunction with the calculate_variables function and uses the kinetic theory of gases to estimate mixed gas properties.

Definition at line 116 of file egret.h.

5.52.2 Member Data Documentation

5.52.2.1 Matrix<double> MIXED_GAS::binary_diffusion

Calculated: Tensor matrix of binary gas diffusivities (cm^2/s)

Definition at line 136 of file egret.h.

5.52.2.2 double MIXED_GAS::char_length

Given: Characteristic Length (cm)

Definition at line 126 of file egret.h.

5.52.2.3 bool MIXED_GAS::CheckMolefractions = true

Given: True = Check Molefractions for errors.

Definition at line 120 of file egret.h.

5.52.2.4 double MIXED_GAS::gas_temperature

Given: Gas temperature (K)

Definition at line 124 of file egret.h.

5.52.2.5 double MIXED_GAS::kinematic_viscosity

Calculated: Kinematic viscosity (cm^2/s)

Definition at line 132 of file egret.h.

5.52.2.6 std::vector<double> MIXED_GAS::molefraction

Given: Gas molefractions of each species (-)

Definition at line 127 of file egret.h.

5.52.2.7 int MIXED_GAS::N

Given: Total number of gas species.

Definition at line 119 of file egret.h.

5.52.2.8 double MIXED_GAS::Reynolds

Calculated: Value of the Reynold's number (-)

Definition at line 135 of file egret.h.

5.52.2.9 std::vector<PURE_GAS> MIXED_GAS::species_dat

Vector of the pure gas info of all species.

Definition at line 139 of file egret.h.

5.52.2.10 double MIXED_GAS::total_density

Calculated: Total gas density (g/cm^3) {use RE3}.

Definition at line 130 of file egret.h.

5.52.2.11 double MIXED_GAS::total_dyn_vis

Calculated: Total dynamic viscosity (g/cm/s)

Definition at line 131 of file egret.h.

5.52.2.12 double MIXED_GAS::total_molecular_weight

Calculated: Total molecular weight (g/mol)

Definition at line 133 of file egret.h.

5.52.2.13 double MIXED_GAS::total_pressure

Given: Total gas pressure (kPa)

Definition at line 123 of file egret.h.

5.52.2.14 double MIXED_GAS::total_specific_heat

Calculated: Total specific heat (J/g/K)

Definition at line 134 of file egret.h.

5.52.2.15 double MIXED_GAS::velocity

Given: Gas phase velocity (cm/s)

Definition at line 125 of file egret.h.

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

- egret.h

5.53 mSPD_DATA Struct Reference

MSPD Data Structure.

`#include <magpie.h>`

Public Attributes

- double s
  Area shape factor.
- double v
  van der Waals Volume (cm^3/mol)
- double eMax
  Maximum lateral interaction energy (J/mol)
- std::vector< double > eta
  Binary interaction parameter matrix (i,j)
- double gama
  Activity coefficient calculated from mSPD.

5.53.1 Detailed Description

MSPD Data Structure.

C-Style object holding all parameter information associated with the Modified Spreading Pressure Dependent (SPD) activity model. Each species in the gas phase will have one of these objects.

Definition at line 110 of file magpie.h.

5.53.2 Member Data Documentation

5.53.2.1 double mSPD_DATA::eMax

Maximum lateral interaction energy (J/mol)

Definition at line 114 of file magpie.h.

5.53.2.2 std::vector<double> mSPD_DATA::eta

Binary interaction parameter matrix (i,j)

Definition at line 115 of file magpie.h.

5.53.2.3 double mSPD_DATA::gama

Activity coefficient calculated from mSPD.

Definition at line 116 of file magpie.h.

5.53.2.4 double mSPD_DATA::s

Area shape factor.

Definition at line 112 of file magpie.h.

5.53.2.5 double mSPD_DATA::v

van der Waals Volume (cm^3/mol)

Definition at line 113 of file magpie.h.

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

- magpie.h

5.54 NUM_JAC_DATA Struct Reference

Data structure to form a numerical jacobian matrix with finite differences.

`#include <lark.h>`

Public Attributes

- double eps = sqrt(DBL_EPSILON)

  Perturbation value.
- Matrix< double > Fx

  Vector of function evaluations at x.
- Matrix< double > Fxp

  Vector of function evaluations at x+eps.
- Matrix< double > dxj

  Vector of perturbed x values.

5.54.1 Detailed Description

Data structure to form a numerical jacobian matrix with finite differences.

C-style object to be used in conjunction with the Numerical Jacobian algorithm. This algorithm will used double-precision finite-differences to formulate an approximate Jacobian matrix at the given variable state for the given residual/non-linear function.

Definition at line 569 of file lark.h.

5.54.2 Member Data Documentation

5.54.2.1 Matrix<double> NUM_JAC_DATA::dxj

Vector of perturbed x values.

Definition at line 574 of file lark.h.

5.54.2.2 double NUM_JAC_DATA::eps = sqrt(DBL_EPSILON)

Perturbation value.

Definition at line 571 of file lark.h.

5.54.2.3 Matrix<double> NUM_JAC_DATA::Fx

Vector of function evaluations at x.

Definition at line 572 of file lark.h.

5.54.2.4 Matrix<double> NUM_JAC_DATA::Fxp

Vector of function evaluations at x+eps.

Definition at line 573 of file lark.h.

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

- lark.h

5.55 OPTRANS_DATA Struct Reference

Data structure for implementation of linear operator transposition.

`#include <lark.h>`

Public Attributes

- Matrix< double > Ii

  The ith column vector of the identity operator.
- Matrix< double > Ai

  The ith column vector of the user's linear operator.

5.55.1 Detailed Description

Data structure for implementation of linear operator transposition.

C-style object used in conjunction with the Operator Transpose algorithm to form an action of A^T*r when A is only available as a linear operator and not a matrix. This is a sub-routine required by GCR and GMRESR to stabilize the outer iterations.

Definition at line 324 of file lark.h.

5.55.2 Member Data Documentation

5.55.2.1 Matrix<double> OPTRANS_DATA::Ai

The ith column vector of the user's linear operator.

Definition at line 327 of file lark.h.

5.55.2.2 Matrix<double> OPTRANS_DATA::Ii

The ith column vector of the identity operator.

Definition at line 326 of file lark.h.

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

- lark.h

5.56 PCG_DATA Struct Reference

Data structure for implementation of the PCG algorithms for symmetric linear systems.

`#include <lark.h>`

Public Attributes

- int maxit = 0

  Maximum allowable iterations - default = min(vector_size,1000)
- int iter = 0

  Actual number of iterations taken.
- double alpha

  Step size for new solution.
- double beta

  Step size for new search direction.

- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm.
- double relres

  Relative residual norm.
- double relres_base

  Initial residual norm.
- double bestres

  Best found residual norm.
- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual vector for the linear system.
- Matrix< double > r_old

  Previous residual vector.
- Matrix< double > z

  Preconditioned residual vector (result of precon function)
- Matrix< double > z_old

  Previous preconditioned residual vector.
- Matrix< double > p

  Search direction.
- Matrix< double > Ap

  Result of matrix-vector multiplication.

5.56.1 Detailed Description

Data structure for implementation of the PCG algorithms for symmetric linear systems.

C-style object used in conjunction with the Preconditioned Conjugate Gradient (PCG) algorithm to iteratively solve a symmetric linear system of equations. This algorithm is optimal if your linear system is symmetric, but will not work at all if your system is asymmetric. For asymmetric systems, use one of the other linear methods.

Definition at line 217 of file lark.h.

5.56.2 Member Data Documentation

5.56.2.1 double PCG_DATA::alpha

Step size for new solution.

Definition at line 222 of file lark.h.

5.56.2.2 Matrix<double> PCG_DATA::Ap

Result of matrix-vector multiplication.

Definition at line 240 of file lark.h.

5.56.2.3 double PCG_DATA::bestres

Best found residual norm.

Definition at line 229 of file lark.h.

5.56.2.4 Matrix<double> PCG_DATA::bestx

Best found solution to the linear system.

Definition at line 234 of file lark.h.

5.56.2.5 double PCG_DATA::beta

Step size for new search direction.

Definition at line 223 of file lark.h.

5.56.2.6 int PCG_DATA::iter = 0

Actual number of iterations taken.

Definition at line 220 of file lark.h.

5.56.2.7 int PCG_DATA::maxit = 0

Maximum allowable iterations - default = min(vector_size,1000)

Definition at line 219 of file lark.h.

5.56.2.8 bool PCG_DATA::Output = true

True = print messages to console.

Definition at line 231 of file lark.h.

5.56.2.9 Matrix<double> PCG_DATA::p

Search direction.

Definition at line 239 of file lark.h.

5.56.2.10 Matrix<double> PCG_DATA::r

Residual vector for the linear system.

Definition at line 235 of file lark.h.

5.56.2.11 Matrix<double> PCG_DATA::r_old

Previous residual vector.

Definition at line 236 of file lark.h.

5.56.2.12 double PCG_DATA::relres

Relative residual norm.

Definition at line 227 of file lark.h.

5.56.2.13 double PCG_DATA::relres_base

Initial residual norm.

Definition at line 228 of file lark.h.

5.56.2.14 double PCG_DATA::res

Absolute residual norm.

Definition at line 226 of file lark.h.

5.56.2.15 double PCG_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 225 of file lark.h.

5.56.2.16 double PCG_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 224 of file lark.h.

5.56.2.17 Matrix<double> PCG_DATA::x

Current solution to the linear system.

Definition at line 233 of file lark.h.

5.56.2.18 Matrix<double> PCG_DATA::z

Preconditioned residual vector (result of precon function)

Definition at line 237 of file lark.h.

5.56.2.19 Matrix<double> PCG_DATA::z_old

Previous preconditioned residual vector.

Definition at line 238 of file lark.h.

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

- lark.h

5.57 PICARD_DATA Struct Reference

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

`#include <lark.h>`

Public Attributes

- int maxit = 0
  Maximum allowable iterations - default = min(3∗vec_size,1000)
- int iter = 0
  Actual number of iterations.
- double tol_rel = 1e-6
  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6
  Absolution tolerance for convergence - default = 1e-6.
- double res
  Residual norm of the iterate.
- double relres
  Relative residual norm of the iterate.
- double relres_base

Initial residual norm.

- double bestres

  Best found residual norm.

- bool Output = true

  True = print messages to console.

- Matrix< double > x0

  Previous iterate solution vector.

- Matrix< double > bestx

  Best found solution vector.

- Matrix< double > r

  Residual of the non-linear system.

5.57.1 Detailed Description

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

C-style object used in conjunction with the Picard algorithm for solving a non-linear system of equations. This is an extradorinarily simple iterative method by which a weak or loose form of the non-linear system is solved based on an initial guess. User must supplied a residual function for the non-linear system and a function representing the weak solution. Generally, this method is less efficient than Newton methods, but is significantly cheaper.

Definition at line 449 of file lark.h.

5.57.2 Member Data Documentation

5.57.2.1 double PICARD_DATA::bestres

Best found residual norm.

Definition at line 459 of file lark.h.

5.57.2.2 Matrix<double> PICARD_DATA::bestx

Best found solution vector.

Definition at line 464 of file lark.h.

5.57.2.3 int PICARD_DATA::iter = 0

Actual number of iterations.

Definition at line 452 of file lark.h.

5.57.2.4 int PICARD_DATA::maxit = 0

Maximum allowable iterations - default = min(3∗vec_size,1000)

Definition at line 451 of file lark.h.

5.57.2.5 bool PICARD_DATA::Output = true

True = print messages to console.

Definition at line 461 of file lark.h.

5.57.2.6 Matrix<double> PICARD_DATA::r

Residual of the non-linear system.

Definition at line 465 of file lark.h.

5.57.2.7 double PICARD_DATA::relres

Relative residual norm of the iterate.

Definition at line 457 of file lark.h.

5.57.2.8 double PICARD_DATA::relres_base

Initial residual norm.

Definition at line 458 of file lark.h.

5.57.2.9 double PICARD_DATA::res

Residual norm of the iterate.

Definition at line 456 of file lark.h.

5.57.2.10 double PICARD_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 455 of file lark.h.

5.57.2.11 double PICARD_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 454 of file lark.h.

5.57.2.12 Matrix<double> PICARD_DATA::x0

Previous iterate solution vector.

Definition at line 463 of file lark.h.

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

- lark.h

5.58 PJFNK_DATA Struct Reference

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

`#include <lark.h>`

Public Attributes

- int nl_iter = 0
  Number of non-linear iterations.
- int l_iter = 0
  Number of linear iterations.
- int fun_call = 0
  Actual number of function calls made.
- int nl_maxit = 0
  Maximum allowable non-linear steps.
- int linear_solver = -1
  Flag to denote which linear solver to use - default = PJFNK Chooses.
- double nl_tol_abs = 1e-6
  Absolute Convergence tolerance for non-linear system - default = 1e-6.
- double nl_tol_rel = 1e-6

Relative Convergence tol for the non-linear system - default = 1e-6.

- double lin_tol_rel = 1e-6

  Relative tolerance of the linear solver - default = 1e-6.
- double lin_tol_abs = 1e-6

  Absolute tolerance of the linear solver - default = 1e-6.
- double nl_res

  Absolute redidual norm for the non-linear system.
- double nl_relres

  Relative residual for the non-linear system.
- double nl_res_base

  Initial residual norm for the non-linear system.
- double nl_bestres

  Best found residual norm.
- double eps =sqrt(DBL_EPSILON)

  Value of epsilon used jacvec - default = sqrt(DBL_EPSILON)
- bool NL_Output = true

  True = print PJFNK messages to console.
- bool L_Output = false

  True = print Linear messages to console.
- bool LineSearch = false

  True = use Backtracking Linesearch for global convergence.
- bool Bounce = false

  True = allow Linesearch to go outside local well, False = Strict local convergence.
- Matrix< double > F

  Stored fuction evaluation at x (also the residual)
- Matrix< double > Fv

  Stored function evaluation at x+eps∗v.
- Matrix< double > v

  Stored vector of x+eps∗v.
- Matrix< double > x

  Current solution vector for the non-linear system.
- Matrix< double > bestx

  Best found solution vector to the non-linear system.
- GMRESLP_DATA gmreslp_dat

  Data structure for the GMRESLP method.
- PCG_DATA pcg_dat

  Data structure for the PCG method.
- BiCGSTAB_DATA bicgstab_dat

  Data structure for the BiCGSTAB method.
- CGS_DATA cgs_dat

  Data structure for the CGS method.
- GMRESRP_DATA gmresrp_dat

  Data structure for the GMRESRP method.
- GCR_DATA gcr_dat

  Data structure for the GCR method.
- GMRESR_DATA gmresr_dat

  Data structure for the GMRESR method.
- BACKTRACK_DATA backtrack_dat

  Data structure for the Backtracking Linesearch algorithm.
- const void ∗ res_data

  Data structure pointer for user's residual data.

- const void * precon_data

  Data structure pointer for user's preconditioning data.
- int(* funeval)(const Matrix< double > &x, Matrix< double > &F, const void *res_data)

  Function pointer for the user's function F(x) using there data.
- int(* precon)(const Matrix< double > &r, Matrix< double > &p, const void *precon_data)

  Function pointer for the user's preconditioning function for the linear system.

5.58.1 Detailed Description

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

C-style object to be used in conjunction with the Preconditioned Jacobian-Free Newton-Krylov (PJFNK) method for solving a non-linear system of equations. You can use any of the Krylov methods listed in the krylov_method enum to solve the linear sub-problem. When FOM is specified as the Krylov method, this algorithm becomes equivalent to an exact Newton method. If no Krylov method is specified, then the algorithm will try to pick a method based on the problem size and availability of preconditioning.

Definition at line 511 of file lark.h.

5.58.2 Member Data Documentation

5.58.2.1 BACKTRACK_DATA PJFNK_DATA::backtrack_dat

Data structure for the Backtracking Linesearch algorithm.

Definition at line 550 of file lark.h.

5.58.2.2 Matrix<double> PJFNK_DATA::bestx

Best found solution vector to the non-linear system.

Definition at line 538 of file lark.h.

5.58.2.3 BiCGSTAB_DATA PJFNK_DATA::bicgstab_dat

Data structure for the BiCGSTAB method.

Definition at line 543 of file lark.h.

5.58.2.4 bool PJFNK_DATA::Bounce = false

True = allow Linesearch to go outside local well, False = Strict local convergence.

Definition at line 532 of file lark.h.

5.58.2.5 CGS_DATA PJFNK_DATA::cgs_dat

Data structure for the CGS method.

Definition at line 544 of file lark.h.

5.58.2.6 double PJFNK_DATA::eps =sqrt(DBL_EPSILON)

Value of epsilon used jacvec - default = sqrt(DBL_EPSILON)

Definition at line 527 of file lark.h.

5.58.2.7 Matrix<double> PJFNK_DATA::F

Stored fuction evaluation at x (also the residual)

Definition at line 534 of file lark.h.

5.58.2.8 int PJFNK_DATA::fun_call = 0

Actual number of function calls made.

Definition at line 515 of file lark.h.

5.58.2.9 int(* PJFNK_DATA::funeval)(const Matrix< double > &x, Matrix< double > &F, const void *res_data)

Function pointer for the user's function F(x) using there data.

Definition at line 559 of file lark.h.

5.58.2.10 Matrix<double> PJFNK_DATA::Fv

Stored function evaluation at x+eps*v.

Definition at line 535 of file lark.h.

5.58.2.11 GCR_DATA PJFNK_DATA::gcr_dat

Data structure for the GCR method.

Definition at line 546 of file lark.h.

5.58.2.12 GMRESLP_DATA PJFNK_DATA::gmreslp_dat

Data structure for the GMRESLP method.

Definition at line 541 of file lark.h.

5.58.2.13 GMRESR_DATA PJFNK_DATA::gmresr_dat

Data structure for the GMRESR method.

Definition at line 547 of file lark.h.

5.58.2.14 GMRESRP_DATA PJFNK_DATA::gmresrp_dat

Data structure for the GMRESRP method.

Definition at line 545 of file lark.h.

5.58.2.15 int PJFNK_DATA::l_iter = 0

Number of linear iterations.

Definition at line 514 of file lark.h.

5.58.2.16 bool PJFNK_DATA::L_Output = false

True = print Linear messages to console.

Definition at line 530 of file lark.h.

5.58.2.17 double PJFNK_DATA::lin_tol_abs = 1e-6

Absolute tolerance of the linear solver - default = 1e-6.

Definition at line 522 of file lark.h.

5.58.2.18 double PJFNK_DATA::lin_tol_rel = 1e-6

Relative tolerance of the linear solver - default = 1e-6.

Definition at line 521 of file lark.h.

5.58.2.19 int PJFNK_DATA::linear_solver = -1

Flag to denote which linear solver to use - default = PJFNK Chooses.

Definition at line 517 of file lark.h.

5.58.2.20 bool PJFNK_DATA::LineSearch = false

True = use Backtracking Linesearch for global convergence.

Definition at line 531 of file lark.h.

5.58.2.21 double PJFNK_DATA::nl_bestres

Best found residual norm.

Definition at line 526 of file lark.h.

5.58.2.22 int PJFNK_DATA::nl_iter = 0

Number of non-linear iterations.

Definition at line 513 of file lark.h.

5.58.2.23 int PJFNK_DATA::nl_maxit = 0

Maximum allowable non-linear steps.

Definition at line 516 of file lark.h.

5.58.2.24 bool PJFNK_DATA::NL_Output = true

True = print PJFNK messages to console.

Definition at line 529 of file lark.h.

5.58.2.25 double PJFNK_DATA::nl_relres

Relative residual for the non-linear system.

Definition at line 524 of file lark.h.

5.58.2.26 double PJFNK_DATA::nl_res

Absolute redidual norm for the non-linear system.

Definition at line 523 of file lark.h.

5.58.2.27 double PJFNK_DATA::nl_res_base

Initial residual norm for the non-linear system.

Definition at line 525 of file lark.h.

5.58.2.28 double PJFNK_DATA::nl_tol_abs = 1e-6

Absolute Convergence tolerance for non-linear system - default = 1e-6.

Definition at line 519 of file lark.h.

5.58.2.29 double PJFNK_DATA::nl_tol_rel = 1e-6

Relative Convergence tol for the non-linear system - default = 1e-6.

Definition at line 520 of file lark.h.

5.58.2.30 PCG_DATA PJFNK_DATA::pcg_dat

Data structure for the PCG method.

Definition at line 542 of file lark.h.

5.58.2.31 int(* PJFNK_DATA::precon)(const Matrix< double > &r, Matrix< double > &p, const void *precon_data)

Function pointer for the user's preconditioning function for the linear system.

Definition at line 561 of file lark.h.

5.58.2.32 const void* PJFNK_DATA::precon_data

Data structure pointer for user's preconditioning data.

Definition at line 557 of file lark.h.

5.58.2.33 const void* PJFNK_DATA::res_data

Data structure pointer for user's residual data.

Definition at line 555 of file lark.h.

5.58.2.34 Matrix<double> PJFNK_DATA::v

Stored vector of x+eps*v.

Definition at line 536 of file lark.h.

5.58.2.35 Matrix<double> PJFNK_DATA::x

Current solution vector for the non-linear system.

Definition at line 537 of file lark.h.

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

- lark.h

5.59 PURE_GAS Struct Reference

Data structure holding all the parameters for each pure gas spieces.

`#include <egret.h>`

Public Attributes

- double molecular_weight
  Given: molecular weights (g/mol)
- double Sutherland_Temp
  Given: Sutherland's Reference Temperature (K)
- double Sutherland_Const
  Given: Sutherland's Constant (K)
- double Sutherland_Viscosity
  Given: Sutherland's Reference Viscosity (g/cm/s)
- double specific_heat
  Given: Specific heat of the gas (J/g/K)
- double molecular_diffusion
  Calculated: molecular diffusivities (cm^2/s)
- double dynamic_viscosity

Calculated: dynamic viscosities (g/cm/s)

- double density

  Calculated: gas densities (g/cm^3) {use RE3}.

- double Schmidt

  Calculated: Value of the Schmidt number (-)

5.59.1 Detailed Description

Data structure holding all the parameters for each pure gas spieces.

C-style object that holds the constants and parameters associated with each pure gas species in the overall mixture. This information is used in conjunction with the kinetic theory of gases to produce approximations to many different gas properties needed in simulating gas dynamics, mobility of a gas through porous media, as well as some kinetic adsorption parameters such as diffusivities.

Definition at line 95 of file egret.h.

5.59.2 Member Data Documentation

5.59.2.1 double PURE_GAS::density

Calculated: gas densities (g/cm^3) {use RE3}.

Definition at line 107 of file egret.h.

5.59.2.2 double PURE_GAS::dynamic_viscosity

Calculated: dynamic viscosities (g/cm/s)

Definition at line 106 of file egret.h.

5.59.2.3 double PURE_GAS::molecular_diffusion

Calculated: molecular diffusivities (cm^2/s)

Definition at line 105 of file egret.h.

5.59.2.4 double PURE_GAS::molecular_weight

Given: molecular weights (g/mol)

Definition at line 98 of file egret.h.

5.59.2.5 double PURE_GAS::Schmidt

Calculated: Value of the Schmidt number (-)

Definition at line 108 of file egret.h.

5.59.2.6 double PURE_GAS::specific_heat

Given: Specific heat of the gas (J/g/K)

Definition at line 102 of file egret.h.

5.59.2.7 double PURE_GAS::Sutherland_Const

Given: Sutherland's Constant (K)

Definition at line 100 of file egret.h.

5.59.2.8 double PURE_GAS::Sutherland_Temp

Given: Sutherland's Reference Temperature (K)

Definition at line 99 of file egret.h.

5.59.2.9 double PURE_GAS::Sutherland_Viscosity

Given: Sutherland's Reference Viscosity (g/cm/s)

Definition at line 101 of file egret.h.

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

- egret.h

5.60 SCOPSOWL_DATA Struct Reference

Primary data structure for SCOPSOWL simulations.

`#include <scopsowl.h>`

Public Attributes

- unsigned long int total_steps
  Running total of all calculation steps.
- int coord_macro
  Coordinate system for large pellet.
- int coord_micro
  Coordinate system for small crystal (if any)
- int level = 2
  Level of coupling between the different scales (default = 2)
- double sim_time
  Stopping time for the simulation (hrs)
- double t_old
  Old time of the simulations (hrs)
- double t
  Current time of the simulations (hrs)
- double t_counter = 0.0
  Counter for the time output.
- double t_print
  Print output at every t_print time (hrs)
- bool Print2File = true
  True = results to .txt; False = no printing.
- bool Print2Console = true
  True = results to console; False = no printing.
- bool SurfDiff = true
  True = includes SKUA simulation if Heterogeneous; False = only uses MAGPIE.
- bool Heterogeneous = true
  True = pellet is made of binder and crystals, False = all one phase.
- double gas_velocity
  Superficial Gas Velocity arount pellet (cm/s)
- double total_pressure
  Gas phase total pressure (kPa)

- double gas_temperature

  Gas phase temperature (K)
- double pellet_radius

  Nominal radius of the pellet - macroscale domain (cm)
- double crystal_radius

  Nominal radius of the crystal - microscale domain (um)
- double char_macro

  Characteristic size for macro scale (cm or cm^2) - only if pellet is not spherical.
- double char_micro

  Characteristic size for micro scale (um or um^2) - only if crystal is not spherical.
- double binder_fraction

  Volume of binder per total volume of pellet (-)
- double binder_porosity

  Volume of pores per volume of binder (-)
- double binder_poresize

  Nominal radius of the binder pores (cm)
- double pellet_density

  Mass of the pellet per volume of pellet (kg/L)
- bool DirichletBC = false

  True = Dirichlet BC; False = Neumann BC.
- bool NonLinear = true

  True = Non-linear solver; False = Linear solver.
- std::vector< double > y

  Outside mole fractions of each component (-)
- std::vector< double > tempy

  Temporary place holder for gas mole fractions in other locations (-)
- FILE * OutputFile

  Output file pointer to the output file for postprocesses.
- double(* eval_ads)(int i, int l, const void *user_data)

  Function pointer for evaluating adsorption (mol/kg)
- double(* eval_retard)(int i, int l, const void *user_data)

  Function pointer for evaluating retardation (-)
- double(* eval_diff)(int i, int l, const void *user_data)

  Function pointer for evaluating pore diffusion (cm^2/hr)
- double(* eval_surfDiff)(int i, int l, const void *user_data)

  Function pointer for evaluating surface diffusion (um^2/hr)
- double(* eval_kf)(int i, const void *user_data)

  Function pointer for evaluating film mass transfer (cm/hr)
- const void * user_data

  Data structure for users info to calculate parameters.
- MIXED_GAS * gas_dat

  Pointer to the MIXED_GAS data structure (may or may not be used)
- MAGPIE_DATA magpie_dat

  Data structure for a magpie problem (to be used if not using skua)
- std::vector< FINCH_DATA > finch_dat

  Data structure for pore adsorption kinetics for all species (u in mol/L)
- std::vector< SCOPSOWL_PARAM_DATA > param_dat

  Data structure for parameter info for all species.
- std::vector< SKUA_DATA > skua_dat

  Data structure holding a skua object for all nodes (each skua has an object for each species)

5.60.1 Detailed Description

Primary data structure for SCOPSOWL simulations.

C-style object holding necessary information to run a SCOPSOWL simulation. SCOPSOWL is a multi-scale problem involving PDE solution for the macro-scale adsorbent pellet and the micro-scale adsorbent crystals. As such, each SCOPSOWL simulation involves multiple SKUA simulations at the nodes in the macro-scale domain. Alternatively, if the user wishes to specify that the adsorbent is homogeneous, then you can run SCOPSOWL as a single-scale problem. Additionally, you can simplfy the model by assuming that the micro-scale diffusion is very fast, and therefore replace each SKUA simulation with a simpler MAGPIE evaluation. Details on running SCOPSOWL with the various options will be discussed in the SCOPSOWL_SCENARIOS function.

Definition at line 92 of file scopsowl.h.

5.60.2 Member Data Documentation

5.60.2.1 double SCOPSOWL_DATA::binder_fraction

Volume of binder per total volume of pellet (-)

Definition at line 116 of file scopsowl.h.

5.60.2.2 double SCOPSOWL_DATA::binder_poresize

Nominal radius of the binder pores (cm)

Definition at line 118 of file scopsowl.h.

5.60.2.3 double SCOPSOWL_DATA::binder_porosity

Volume of pores per volume of binder (-)

Definition at line 117 of file scopsowl.h.

5.60.2.4 double SCOPSOWL_DATA::char_macro

Characteristic size for macro scale (cm or cm^2) - only if pellet is not spherical.

Definition at line 114 of file scopsowl.h.

5.60.2.5 double SCOPSOWL_DATA::char_micro

Characteristic size for micro scale (um or um^2) - only if crystal is not spherical.

Definition at line 115 of file scopsowl.h.

5.60.2.6 int SCOPSOWL_DATA::coord_macro

Coordinate system for large pellet.

Definition at line 95 of file scopsowl.h.

5.60.2.7 int SCOPSOWL_DATA::coord_micro

Coordinate system for small crystal (if any)

Definition at line 96 of file scopsowl.h.

5.60.2.8 double SCOPSOWL_DATA::crystal_radius

Nominal radius of the crystal - microscale domain (um)

Definition at line 113 of file scopsowl.h.

5.60.2.9 bool SCOPSOWL_DATA::DirichletBC = false

True = Dirichlet BC; False = Neumann BC.

Definition at line 121 of file scopsowl.h.

5.60.2.10 double(* SCOPSOWL_DATA::eval_ads)(int i, int l, const void *user_data)

Function pointer for evaluating adsorption (mol/kg)

Definition at line 127 of file scopsowl.h.

5.60.2.11 double(* SCOPSOWL_DATA::eval_diff)(int i, int l, const void *user_data)

Function pointer for evaluating pore diffusion (cm^2/hr)

Definition at line 129 of file scopsowl.h.

5.60.2.12 double(* SCOPSOWL_DATA::eval_kf)(int i, const void *user_data)

Function pointer for evaluating film mass transfer (cm/hr)

Definition at line 131 of file scopsowl.h.

5.60.2.13 double(* SCOPSOWL_DATA::eval_retard)(int i, int l, const void *user_data)

Function pointer for evaluating retardation (-)

Definition at line 128 of file scopsowl.h.

5.60.2.14 double(* SCOPSOWL_DATA::eval_surfDiff)(int i, int l, const void *user_data)

Function pointer for evaluating surface diffusion (um^2/hr)

Definition at line 130 of file scopsowl.h.

5.60.2.15 std::vector<FINCH_DATA> SCOPSOWL_DATA::finch_dat

Data structure for pore adsorption kinetics for all species (u in mol/L)

Definition at line 136 of file scopsowl.h.

5.60.2.16 MIXED_GAS* SCOPSOWL_DATA::gas_dat

Pointer to the MIXED_GAS data structure (may or may not be used)

Definition at line 134 of file scopsowl.h.

5.60.2.17 double SCOPSOWL_DATA::gas_temperature

Gas phase temperature (K)

Definition at line 111 of file scopsowl.h.

5.60.2.18 double SCOPSOWL_DATA::gas_velocity

Superficial Gas Velocity arount pellet (cm/s)

Definition at line 109 of file scopsowl.h.

5.60.2.19 bool SCOPSOWL_DATA::Heterogeneous = true

True = pellet is made of binder and crystals, False = all one phase.

Definition at line 107 of file scopsowl.h.

5.60.2.20 int SCOPSOWL_DATA::level = 2

Level of coupling between the different scales (default = 2)

Definition at line 97 of file scopsowl.h.

5.60.2.21 MAGPIE_DATA SCOPSOWL_DATA::magpie_dat

Data structure for a magpie problem (to be used if not using skua)

Definition at line 135 of file scopsowl.h.

5.60.2.22 bool SCOPSOWL_DATA::NonLinear = true

True = Non-linear solver; False = Linear solver.

Definition at line 122 of file scopsowl.h.

5.60.2.23 FILE* SCOPSOWL_DATA::OutputFile

Output file pointer to the output file for postprocesses.

Definition at line 126 of file scopsowl.h.

5.60.2.24 std::vector<SCOPSOWL_PARAM_DATA> SCOPSOWL_DATA::param_dat

Data structure for parameter info for all species.

Definition at line 137 of file scopsowl.h.

5.60.2.25 double SCOPSOWL_DATA::pellet_density

Mass of the pellet per volume of pellet (kg/L)

Definition at line 119 of file scopsowl.h.

5.60.2.26 double SCOPSOWL_DATA::pellet_radius

Nominal radius of the pellet - macroscale domain (cm)

Definition at line 112 of file scopsowl.h.

5.60.2.27 bool SCOPSOWL_DATA::Print2Console = true

True = results to console; False = no printing.

Definition at line 105 of file scopsowl.h.

5.60.2.28 bool SCOPSOWL_DATA::Print2File = true

True = results to .txt; False = no printing.

Definition at line 104 of file scopsowl.h.

5.60.2.29 double SCOPSOWL_DATA::sim_time

Stopping time for the simulation (hrs)

Definition at line 98 of file scopsowl.h.

5.60.2.30 std::vector<SKUA_DATA> SCOPSOWL_DATA::skua_dat

Data structure holding a skua object for all nodes (each skua has an object for each species)

Definition at line 139 of file scopsowl.h.

5.60.2.31 bool SCOPSOWL_DATA::SurfDiff = true

True = includes SKUA simulation if Heterogeneous; False = only uses MAGPIE.

Definition at line 106 of file scopsowl.h.

5.60.2.32 double SCOPSOWL_DATA::t

Current time of the simulations (hrs)

Definition at line 100 of file scopsowl.h.

5.60.2.33 double SCOPSOWL_DATA::t_counter = 0.0

Counter for the time output.

Definition at line 101 of file scopsowl.h.

5.60.2.34 double SCOPSOWL_DATA::t_old

Old time of the simulations (hrs)

Definition at line 99 of file scopsowl.h.

5.60.2.35 double SCOPSOWL_DATA::t_print

Print output at every t_print time (hrs)

Definition at line 102 of file scopsowl.h.

5.60.2.36 std::vector<double> SCOPSOWL_DATA::tempy

Temporary place holder for gas mole fractions in other locations (-)

Definition at line 124 of file scopsowl.h.

5.60.2.37 double SCOPSOWL_DATA::total_pressure

Gas phase total pressure (kPa)

Definition at line 110 of file scopsowl.h.

5.60.2.38 unsigned long int SCOPSOWL_DATA::total_steps

Running total of all calculation steps.

Definition at line 94 of file scopsowl.h.

5.60.2.39 const void* SCOPSOWL_DATA::user_data

Data structure for users info to calculate parameters.

Definition at line 133 of file scopsowl.h.

5.60.2.40 std::vector<double> SCOPSOWL_DATA::y

Outside mole fractions of each component (-)

Definition at line 123 of file scopsowl.h.

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

- scopsowl.h

5.61 SCOPSOWL_PARAM_DATA Struct Reference

Data structure for the species' parameters in SCOPSOWL.

`#include <scopsowl.h>`

Public Attributes

- Matrix< double > qAvg

  Average adsorbed amount for a species at each node (mol/kg)
- Matrix< double > qAvg_old

  Old Average adsorbed amount for a species at each node (mol/kg)
- Matrix< double > Qst

  Heat of adsorption for all nodes (J/mol)
- Matrix< double > Qst_old

  Old Heat of adsorption for all nodes (J/mol)
- Matrix< double > dq_dc

  Storage vector for current adsorption slope/strength (dq/dc) (L/kg)
- double xIC

  Initial conditions for adsorbed molefractions.
- double qIntegralAvg

  Integral average of adsorption over the entire pellet (mol/kg)
- double qIntegralAvg_old

  Old Integral average of adsorption over the entire pellet (mol/kg)
- double QstAvg

  Integral average heat of adsorption (J/mol)
- double QstAvg_old

  Old integral average heat of adsorption (J/mol)
- double qo

  Boundary value of adsorption if using Dirichlet BCs (mol/kg)
- double Qsto

  Boundary value of adsorption heat if using Dirichlet BCs (J/mol)
- double dq_dco

  Boundary value of adsorption slope for Dirichelt BCs (L/kg)
- double pore_diffusion

  Value for constant pore diffusion (cm^2/hr)
- double film_transfer

  Value for constant film mass transfer (cm/hr)
- double activation_energy

  Activation energy for surface diffusion (J/mol)
- double ref_diffusion

  Reference state surface diffusivity (um^2/hr)
- double ref_temperature

  Reference temperature for empirical adjustments (K)
- double affinity

  Affinity parameter used in empirical adjustments (-)
- double ref_pressure
- bool Adsorbable

  True = species can adsorb; False = species cannot adsorb.
- std::string speciesName

  String to hold the name of each species.

5.61.1 Detailed Description

Data structure for the species' parameters in SCOPSOWL.

C-style object that holds information on all species for a particular SCOPSOWL simulation. Initial conditions, kinetic parameters, and interim matrix objects are stored here for use in various SCOSPSOWL functions.

Definition at line 44 of file scopsowl.h.

5.61.2 Member Data Documentation

5.61.2.1 double SCOPSOWL_PARAM_DATA::activation_energy

Activation energy for surface diffusion (J/mol)

Definition at line 69 of file scopsowl.h.

5.61.2.2 bool SCOPSOWL_PARAM_DATA::Adsorbable

True = species can adsorb; False = species cannot adsorb.

Definition at line 75 of file scopsowl.h.

5.61.2.3 double SCOPSOWL_PARAM_DATA::affinity

Affinity parameter used in empirical adjustments (-)

Definition at line 72 of file scopsowl.h.

5.61.2.4 Matrix<double> SCOPSOWL_PARAM_DATA::dq_dc

Storage vector for current adsorption slope/strength (dq/dc) (L/kg)

Definition at line 52 of file scopsowl.h.

5.61.2.5 double SCOPSOWL_PARAM_DATA::dq_dco

Boundary value of adsorption slope for Dirichelt BCs (L/kg)

Definition at line 64 of file scopsowl.h.

5.61.2.6 double SCOPSOWL_PARAM_DATA::film_transfer

Value for constant film mass transfer (cm/hr)

Definition at line 67 of file scopsowl.h.

5.61.2.7 double SCOPSOWL_PARAM_DATA::pore_diffusion

Value for constant pore diffusion (cm^2/hr)

Definition at line 66 of file scopsowl.h.

5.61.2.8 Matrix<double> SCOPSOWL_PARAM_DATA::qAvg

Average adsorbed amount for a species at each node (mol/kg)

Definition at line 46 of file scopsowl.h.

5.61.2.9 Matrix<double> SCOPSOWL_PARAM_DATA::qAvg_old

Old Average adsorbed amount for a species at each node (mol/kg)

Definition at line 47 of file scopsowl.h.

5.61.2.10 double SCOPSOWL_PARAM_DATA::qIntegralAvg

Integral average of adsorption over the entire pellet (mol/kg)

Definition at line 56 of file scopsowl.h.

5.61.2.11 double SCOPSOWL_PARAM_DATA::qIntegralAvg_old

Old Integral average of adsorption over the entire pellet (mol/kg)

Definition at line 57 of file scopsowl.h.

5.61.2.12 double SCOPSOWL_PARAM_DATA::qo

Boundary value of adsorption if using Dirichlet BCs (mol/kg)

Definition at line 62 of file scopsowl.h.

5.61.2.13 Matrix<double> SCOPSOWL_PARAM_DATA::Qst

Heat of adsorption for all nodes (J/mol)

Definition at line 49 of file scopsowl.h.

5.61.2.14 Matrix<double> SCOPSOWL_PARAM_DATA::Qst_old

Old Heat of adsorption for all nodes (J/mol)

Definition at line 50 of file scopsowl.h.

5.61.2.15 double SCOPSOWL_PARAM_DATA::QstAvg

Integral average heat of adsorption (J/mol)

Definition at line 59 of file scopsowl.h.

5.61.2.16 double SCOPSOWL_PARAM_DATA::QstAvg_old

Old integral average heat of adsorption (J/mol)

Definition at line 60 of file scopsowl.h.

5.61.2.17 double SCOPSOWL_PARAM_DATA::Qsto

Boundary value of adsorption heat if using Dirichlet BCs (J/mol)

Definition at line 63 of file scopsowl.h.

5.61.2.18 double SCOPSOWL_PARAM_DATA::ref_diffusion

Reference state surface diffusivity (um^2/hr)

Definition at line 70 of file scopsowl.h.

5.61.2.19 double SCOPSOWL_PARAM_DATA::ref_pressure

Definition at line 73 of file scopsowl.h.

5.61.2.20 double SCOPSOWL_PARAM_DATA::ref_temperature

Reference temperature for empirical adjustments (K)

Definition at line 71 of file scopsowl.h.

5.61.2.21 std::string SCOPSOWL_PARAM_DATA::speciesName

String to hold the name of each species.

Definition at line 77 of file scopsowl.h.

5.61.2.22 double SCOPSOWL_PARAM_DATA::xIC

Initial conditions for adsorbed molefractions.

Definition at line 54 of file scopsowl.h.

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

- scopsowl.h

5.62 SKUA_DATA Struct Reference

Data structure for all simulation information in SKUA.

`#include <skua.h>`

Public Attributes

- unsigned long int total_steps
  Running total of all calculation steps.
- int coord
  Used to determine the coordinates of the problem.
- double sim_time
  Stopping time for the simulation (hrs)
- double t_old
  Old time of the simulations (hrs)
- double t
  Current time of the simulations (hrs)
- double t_counter = 0.0
  Counts for print times for output (hrs)
- double t_print
  Prints out every t_print time (hrs)
- double qTn
  Old total amounts adsorbed (mol/kg)
- double qTnp1
  New total amounts adsorbed (mol/kg)
- bool Print2File = true
  True = results to .txt; False = no printing.
- bool Print2Console = true
  True = results to console; False = no printing.
- double gas_velocity
  Superficial Gas Velocity arount pellet (cm/s)
- double pellet_radius
  Nominal radius of the pellet/crystal (um)
- double char_measure
  Length or Area if in Cylindrical or Cartesian coordinates (um or um^2)
- bool DirichletBC = true
  True = Dirichlet BC; False = Neumann BC.
- bool NonLinear = true
  True = Non-linear solver; False = Linear solver.
- std::vector< double > y

Outside mole fractions of each component (-)

- FILE * OutputFile

  Output file pointer to the output file.

- double(* eval_diff)(int i, int l, const void *user_data)

  Function pointer for evaluating surface diffusivity.

- double(* eval_kf)(int i, const void *user_data)

  Function pointer for evaluating film mass transfer.

- const void * user_data

  Data structure for user's information needed in parameter functions.

- MAGPIE_DATA magpie_dat

  Data structure for adsorption equilibria (see magpie.h)

- MIXED_GAS * gas_dat

  Pointer to the MIXED_GAS data structure (see egret.h)

- std::vector< FINCH_DATA > finch_dat

  Data structure for adsorption kinetics (see finch.h)

- std::vector< SKUA_PARAM > param_dat

  Data structure for SKUA specific parameters.

5.62.1 Detailed Description

Data structure for all simulation information in SKUA.

C-style object holding all data, functions, and other objects needed to successfully run a SKUA simulation. This object holds system information, such as boundary condition type, adsorbent size, and total adsorption, and also contains structure for EGRET (egret.h), FINCH (finch.h), and MAGPIE (magpie.h) calculations. Function pointers for evaluation of the surface diffusivity and film mass transfer coefficients can be overriden by the user to change the behavior of the SKUA simulation. However, defaults are also provided for these functions.

Definition at line 84 of file skua.h.

5.62.2 Member Data Documentation

5.62.2.1 double SKUA_DATA::char_measure

Length or Area if in Cylindrical or Cartesian coordinates (um or um^2)

Definition at line 100 of file skua.h.

5.62.2.2 int SKUA_DATA::coord

Used to determine the coordinates of the problem.

Definition at line 87 of file skua.h.

5.62.2.3 bool SKUA_DATA::DirichletBC = true

True = Dirichlet BC; False = Neumann BC.

Definition at line 101 of file skua.h.

5.62.2.4 double(* SKUA_DATA::eval_diff)(int i, int l, const void *user_data)

Function pointer for evaluating surface diffusivity.

Definition at line 106 of file skua.h.

5.62.2.5 double(* SKUA_DATA::eval_kf)(int i, const void *user_data)

Function pointer for evaluating film mass transfer.

Definition at line 107 of file skua.h.

5.62.2.6 std::vector<FINCH_DATA> SKUA_DATA::finch_dat

Data structure for adsorption kinetics (see finch.h)

Definition at line 111 of file skua.h.

5.62.2.7 MIXED_GAS* SKUA_DATA::gas_dat

Pointer to the MIXED_GAS data structure (see egret.h)

Definition at line 110 of file skua.h.

5.62.2.8 double SKUA_DATA::gas_velocity

Superficial Gas Velocity arount pellet (cm/s)

Definition at line 98 of file skua.h.

5.62.2.9 MAGPIE_DATA SKUA_DATA::magpie_dat

Data structure for adsorption equilibria (see magpie.h)

Definition at line 109 of file skua.h.

5.62.2.10 bool SKUA_DATA::NonLinear = true

True = Non-linear solver; False = Linear solver.

Definition at line 102 of file skua.h.

5.62.2.11 FILE* SKUA_DATA::OutputFile

Output file pointer to the output file.

Definition at line 105 of file skua.h.

5.62.2.12 std::vector<SKUA_PARAM> SKUA_DATA::param_dat

Data structure for SKUA specific parameters.

Definition at line 112 of file skua.h.

5.62.2.13 double SKUA_DATA::pellet_radius

Nominal radius of the pellet/crystal (um)

Definition at line 99 of file skua.h.

5.62.2.14 bool SKUA_DATA::Print2Console = true

True = results to console; False = no printing.

Definition at line 96 of file skua.h.

5.62.2.15 bool SKUA_DATA::Print2File = true

True = results to .txt; False = no printing.

Definition at line 95 of file skua.h.

5.62.2.16 double SKUA_DATA::qTn

Old total amounts adsorbed (mol/kg)

Definition at line 93 of file skua.h.

5.62.2.17 double SKUA_DATA::qTnp1

New total amounts adsorbed (mol/kg)

Definition at line 94 of file skua.h.

5.62.2.18 double SKUA_DATA::sim_time

Stopping time for the simulation (hrs)

Definition at line 88 of file skua.h.

5.62.2.19 double SKUA_DATA::t

Current time of the simulations (hrs)

Definition at line 90 of file skua.h.

5.62.2.20 double SKUA_DATA::t_counter = 0.0

Counts for print times for output (hrs)

Definition at line 91 of file skua.h.

5.62.2.21 double SKUA_DATA::t_old

Old time of the simulations (hrs)

Definition at line 89 of file skua.h.

5.62.2.22 double SKUA_DATA::t_print

Prints out every t_print time (hrs)

Definition at line 92 of file skua.h.

5.62.2.23 unsigned long int SKUA_DATA::total_steps

Running total of all calculation steps.

Definition at line 86 of file skua.h.

5.62.2.24 const void* SKUA_DATA::user_data

Data structure for user's information needed in parameter functions.

Definition at line 108 of file skua.h.

5.62.2.25 std::vector<double> SKUA_DATA::y

Outside mole fractions of each component (-)

Definition at line 103 of file skua.h.

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

- skua.h

5.63 SKUA_PARAM Struct Reference

Data structure for species' parameters in SKUA.

`#include <skua.h>`

Public Attributes

- double activation_energy
- double ref_diffusion
- double ref_temperature
- double affinity
- double ref_pressure
- double film_transfer
- double xIC
- double y_eff
- double Qstn
- double Qstnp1
- double xn
- double xnp1
- bool Adsorbable
- std::string speciesName

5.63.1 Detailed Description

Data structure for species' parameters in SKUA.

C-style object holding data and parameters associated with the gas/solid species in the overall SKUA system. These parameters are used in to modify surface diffusivity with temperature, establish film mass transfer coefficients, formulate the initial conditions, and store solution results for heat of adsorption and adsorbed mole fractions. One of these objects will be created for each species in the gas system.

Definition at line 56 of file skua.h.

5.63.2 Member Data Documentation

5.63.2.1 double SKUA_PARAM::activation_energy

Definition at line 58 of file skua.h.

5.63.2.2 bool SKUA_PARAM::Adsorbable

Definition at line 73 of file skua.h.

5.63.2.3 double SKUA_PARAM::affinity

Definition at line 61 of file skua.h.

5.63.2.4 double SKUA_PARAM::film_transfer

Definition at line 63 of file skua.h.

5.63.2.5 double SKUA_PARAM::Qstn

Definition at line 68 of file skua.h.

5.63.2.6 double SKUA_PARAM::Qstnp1

Definition at line 69 of file skua.h.

5.63.2.7 double SKUA_PARAM::ref_diffusion

Definition at line 59 of file skua.h.

5.63.2.8 double SKUA_PARAM::ref_pressure

Definition at line 62 of file skua.h.

5.63.2.9 double SKUA_PARAM::ref_temperature

Definition at line 60 of file skua.h.

5.63.2.10 std::string SKUA_PARAM::speciesName

Definition at line 75 of file skua.h.

5.63.2.11 double SKUA_PARAM::xIC

Definition at line 65 of file skua.h.

5.63.2.12 double SKUA_PARAM::xn

Definition at line 70 of file skua.h.

5.63.2.13 double SKUA_PARAM::xnp1

Definition at line 71 of file skua.h.

5.63.2.14 double SKUA_PARAM::y_eff

Definition at line 66 of file skua.h.

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

- skua.h

5.64 SYSTEM_DATA Struct Reference

System Data Structure.

`#include <magpie.h>`

Public Attributes

- double T
  System Temperature (K)
- double PT
  Total Pressure (kPa)
- double qT
  Total Amount adsorbed (mol/kg)
- double PI
  Total Lumped Spreading Pressure (mol/kg)
- double pi
  Actual Spreading pressure (J/m^2)
- double As
  Specific surface area of adsorbent (m^2/kg)
- int N
  Total Number of Components.

- int I
- int J
- int K

  Special indices used to keep track of sub-systems.
- unsigned long int total_eval

  Counter to keep track of total number of non-linear steps.
- double avg_norm

  Used to store all norms from evaluations then average at end of run.
- double max_norm

  Used to store the maximum e.norm calculated from non-linear iterations.
- int Sys

  Number of sub-systems to solve.
- int Par

  Number of binary parameters to solve for.
- bool Recover

  If Recover == false, standard GPAST using y's as knowns.
- bool Carrier

  If there is an inert carrier gas, Carrier == true.
- bool Ideal

  If the behavior of the system is determined to be ideal, then Ideal == true.
- bool Output

  Boolean to suppress output if desired (true = display, false = no display.

5.64.1 Detailed Description

System Data Structure.

C-style object holding all the data associated with the overall system to be modeled.

Definition at line 139 of file magpie.h.

5.64.2 Member Data Documentation

5.64.2.1 double SYSTEM_DATA::As

Specific surface area of adsorbent (m^2/kg)

Definition at line 146 of file magpie.h.

5.64.2.2 double SYSTEM_DATA::avg_norm

Used to store all norms from evaluations then average at end of run.

Definition at line 150 of file magpie.h.

5.64.2.3 bool SYSTEM_DATA::Carrier

If there is an inert carrier gas, Carrier == true.

Definition at line 155 of file magpie.h.

5.64.2.4 int SYSTEM_DATA::I

Definition at line 148 of file magpie.h.

5.64.2.5 bool SYSTEM_DATA::Ideal

If the behavior of the system is determined to be ideal, then Ideal == true.

Definition at line 156 of file magpie.h.

5.64.2.6 int SYSTEM_DATA::J

Definition at line 148 of file magpie.h.

5.64.2.7 int SYSTEM_DATA::K

Special indices used to keep track of sub-systems.

Definition at line 148 of file magpie.h.

5.64.2.8 double SYSTEM_DATA::max_norm

Used to store the maximum e.norm calculated from non-linear iterations.

Definition at line 151 of file magpie.h.

5.64.2.9 int SYSTEM_DATA::N

Total Number of Components.

Definition at line 147 of file magpie.h.

5.64.2.10 bool SYSTEM_DATA::Output

Boolean to suppress output if desired (true = display, false = no display.

Definition at line 157 of file magpie.h.

5.64.2.11 int SYSTEM_DATA::Par

Number of binary parameters to solve for.

Definition at line 153 of file magpie.h.

5.64.2.12 double SYSTEM_DATA::Pl

Total Lumped Spreading Pressure (mol/kg)

Definition at line 144 of file magpie.h.

5.64.2.13 double SYSTEM_DATA::pi

Actual Spreading pressure (J/m^2)

Definition at line 145 of file magpie.h.

5.64.2.14 double SYSTEM_DATA::PT

Total Pressure (kPa)

Definition at line 142 of file magpie.h.

5.64.2.15 double SYSTEM_DATA::qT

Total Amount adsorbed (mol/kg)

Definition at line 143 of file magpie.h.

5.64.2.16 bool SYSTEM_DATA::Recover

If Recover == false, standard GPAST using y's as knowns.

Definition at line 154 of file magpie.h.

5.64.2.17 int SYSTEM_DATA::Sys

Number of sub-systems to solve.

Definition at line 152 of file magpie.h.

5.64.2.18 double SYSTEM_DATA::T

System Temperature (K)

Definition at line 141 of file magpie.h.

5.64.2.19 unsigned long int SYSTEM_DATA::total_eval

Counter to keep track of total number of non-linear steps.

Definition at line 149 of file magpie.h.

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

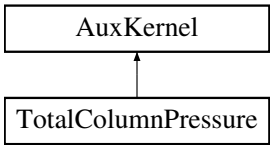
- magpie.h

5.65 TotalColumnPressure Class Reference

Total Column Pressure class inherits from AuxKernel.

`#include <TotalColumnPressure.h>`

Inheritance diagram for TotalColumnPressure:



Public Member Functions

- TotalColumnPressure (const InputParameters &parameters)

  Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

  Required MOOSE function override.

Private Attributes

- VariableValue & _temperature

  Reference to the temperature non-linear variable.
- std::vector< unsigned int > _index

  Indices of the gaseous species coupled to the object.
- std::vector< VariableValue * > _gas_conc

  Pointer list for the non-linear concentration variables.

5.65.1 Detailed Description

Total Column Pressure class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the total column pressure (in kPa) based on the non-linear variables of temperature and concentration of each species in the gas phase. Total pressure is calculated based on the ideal gas law.

Definition at line 54 of file TotalColumnPressure.h.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 TotalColumnPressure::TotalColumnPressure (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.65.3 Member Function Documentation

5.65.3.1 virtual Real TotalColumnPressure::computeValue ( ) [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the total system pressure is needed. You are required to override this function for any inherited AuxKernel.

5.65.4 Member Data Documentation

5.65.4.1 std::vector<VariableValue *> TotalColumnPressure::_gas_conc [private]

Pointer list for the non-linear concentration variables.

Definition at line 70 of file TotalColumnPressure.h.

5.65.4.2 std::vector<unsigned int> TotalColumnPressure::_index [private]

Indices of the gaseous species coupled to the object.

Definition at line 69 of file TotalColumnPressure.h.

5.65.4.3 VariableValue& TotalColumnPressure::_temperature [private]

Reference to the temperature non-linear variable.

Definition at line 68 of file TotalColumnPressure.h.

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

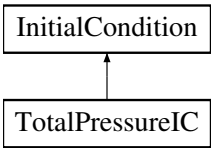
- TotalColumnPressure.h

5.66 TotalPressureIC Class Reference

TotalPressureIC class object inherits from InitialCondition object.

`#include <TotalPressureIC.h>`

Inheritance diagram for TotalPressureIC:



Public Member Functions

- TotalPressureIC (const InputParameters &parameters)

  Required constructor for objects in MOOSE.
- virtual Real value (const Point &p)

  Required function override for setting the value of the non-linear variable at a given point.

Private Attributes

- Real _PT_IC

  Initial condition for the total pressure in the column (kPa)

5.66.1 Detailed Description

TotalPressureIC class object inherits from InitialCondition object.

This class object inherits from the InitialCondition object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will establish the initial conditions for total pressure as constant throughout the domain.

Note

You can have the non-linear variable vary spatially in the domain by inheriting from and or modifying this file to do so.

Definition at line 58 of file TotalPressureIC.h.

5.66.2 Constructor & Destructor Documentation

5.66.2.1 TotalPressureIC::TotalPressureIC (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.66.3 Member Function Documentation

5.66.3.1 virtual Real TotalPressureIC::value (const Point & *p*) [virtual]

Required function override for setting the value of the non-linear variable at a given point.

This function passes a point p as an argument. The return value will be the value of the non-linear variable at that point. That information is used to establish the spatially varying initial condition for the given non-linear variable.

5.66.4 Member Data Documentation

5.66.4.1 Real TotalPressureIC::_PT_IC [private]

Initial condition for the total pressure in the column (kPa)

Definition at line 70 of file TotalPressureIC.h.

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

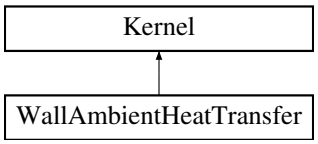
- TotalPressureIC.h

5.67 WallAmbientHeatTransfer Class Reference

WallAmbientHeatTransfer class object inherits from Kernel object.

`#include <WallAmbientHeatTransfer.h>`

Inheritance diagram for WallAmbientHeatTransfer:



Public Member Functions

- WallAmbientHeatTransfer (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- const MaterialProperty< Real > & _wall_exterior_transfer_coeff

  Reference to the wall-exterior heat transfer material property.
- const MaterialProperty< Real > & _inner_dia

  Reference to the wall inner diameter material property.
- const MaterialProperty< Real > & _outer_dia

  Reference to the wall outer diameter material property.
- VariableValue & _ambient_temp

  Reference to the outside temperature coupled non-linear variable.

5.67.1 Detailed Description

WallAmbientHeatTransfer class object inherits from Kernel object.

This class object inherits from the Kernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The kernel interfaces the material properties for the size of the column, as well as the heat transfer coefficient for the exchange of energy from the exterior to the wall, in order to form a residuals and Jacobians for the wall temperature variable.

Definition at line 56 of file WallAmbientHeatTransfer.h.

5.67.2 Constructor & Destructor Documentation

5.67.2.1 WallAmbientHeatTransfer::WallAmbientHeatTransfer (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.67.3 Member Function Documentation

5.67.3.1 virtual Real WallAmbientHeatTransfer::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.67.3.2 virtual Real WallAmbientHeatTransfer::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.67.4 Member Data Documentation

5.67.4.1 VariableValue& WallAmbientHeatTransfer::_ambient_temp [private]

Reference to the outside temperature coupled non-linear variable.

Definition at line 77 of file WallAmbientHeatTransfer.h.

5.67.4.2 const MaterialProperty<Real>& WallAmbientHeatTransfer::_inner_dia [private]

Reference to the wall inner diameter material property.

Definition at line 74 of file WallAmbientHeatTransfer.h.

5.67.4.3 const MaterialProperty<Real>& WallAmbientHeatTransfer::_outer_dia [private]

Reference to the wall outer diameter material property.

Definition at line 75 of file WallAmbientHeatTransfer.h.

5.67.4.4 const MaterialProperty<Real>& WallAmbientHeatTransfer::_wall_exterior_transfer_coeff [private]

Reference to the wall-exterior heat transfer material property.

Definition at line 73 of file WallAmbientHeatTransfer.h.

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

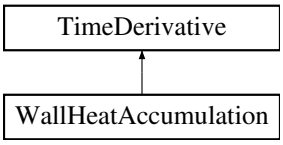
- WallAmbientHeatTransfer.h

5.68 WallHeatAccumulation Class Reference

WallHeatAccumulation class object inherits from TimeDerivative object.

`#include <WallHeatAccumulation.h>`

Inheritance diagram for WallHeatAccumulation:



Public Member Functions

- WallHeatAccumulation (const InputParameters &parameters)

  Required constructor for objects in MOOSE.

Protected Member Functions

- virtual Real computeQpResidual ()

  Required residual function for standard kernels in MOOSE.
- virtual Real computeQpJacobian ()

  Required Jacobian function for standard kernels in MOOSE.

Private Attributes

- const MaterialProperty< Real > & _wall_density

  Reference to the wall density material property.
- const MaterialProperty< Real > & _wall_heat_capacity

  Reference to the wall heat capacity material property.

5.68.1 Detailed Description

WallHeatAccumulation class object inherits from TimeDerivative object.

This class object inherits from the TimeDerivative object. All public and protected members of this class are required function overrides. The kernel interfaces with the wall density and wall heat capacity parameters to generate an time derivative kernel for how the heat in the wall changes based on material density and thermal capacity.

Definition at line 54 of file WallHeatAccumulation.h.

5.68.2 Constructor & Destructor Documentation

5.68.2.1 WallHeatAccumulation::WallHeatAccumulation (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.68.3 Member Function Documentation

5.68.3.1 virtual Real WallHeatAccumulation::computeQpJacobian ( ) [protected], [virtual]

Required Jacobian function for standard kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

5.68.3.2 virtual Real WallHeatAccumulation::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.68.4 Member Data Documentation

5.68.4.1 const MaterialProperty<Real>& WallHeatAccumulation::_wall_density [private]

Reference to the wall density material property.

Definition at line 71 of file WallHeatAccumulation.h.

5.68.4.2 const MaterialProperty<Real>& WallHeatAccumulation::_wall_heat_capacity [private]

Reference to the wall heat capacity material property.

Definition at line 72 of file WallHeatAccumulation.h.

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

- WallHeatAccumulation.h

6 File Documentation

6.1 AdsorbentProperties.h File Reference

Material Properties kernel that will setup and hold all information associated with the adsorbent.

```
#include "Material.h"
```

Classes

- class AdsorbentProperties

  AdsorbentProperties class object inherits from Material object.

Functions

- template<>
  InputParameters validParams< AdsorbentProperties > ()

6.1.1 Detailed Description

Material Properties kernel that will setup and hold all information associated with the adsorbent. Material Properties kernel that will setup and hold all information associated with the fixed-bed.

This file creates a material property object for various properties of a given adsorbent. These properties are then used in other material property files and/or kernels to calculate information such as linear velocity, mechanical dispersion, or any adsorption kinetic parameters.

Note

Currently, we do not couple with adsorption kinetics, so this file is only used in conjunction with the linear velocity and mechanical dispersion properties.

Warning

THIS KERNEL IS INCOMPLETE! ONLY USED FOR DATA STORAGE FOR PELLET DENSITY AND HEAT CAPACITY!

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This file creates a material property object for various properties of the fixed bed. Those properties are used in conjunction with other kernels and materials to establish information such as heat transfer coefficients, conductivities, and size parameters.

Warning

THIS KERNEL IS INCOMPLETE! ONLY USED FOR DATA STORAGE FOR VARIOUS BED PARAMETERS!

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file AdsorbentProperties.h.

6.1.2 Function Documentation

6.1.2.1 template<> InputParameters validParams< AdsorbentProperties >( )

6.2 AdsorptionHeatAccumulation.h File Reference

Standard kernel for the heat of adsorption and its effect on the system temperature.

```
#include "Kernel.h"
```

Classes

- class AdsorptionHeatAccumulation

  AdsorptionHeatAccumulation class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< AdsorptionHeatAccumulation > ()

6.2.1 Detailed Description

Standard kernel for the heat of adsorption and its effect on the system temperature. This file creates a standard MOOSE kernel for the transfer of energy as heat between the bulk gas temperature of the fixed-bed and the adsorbent material in the column. The heat transfer is based on the heat of adsorption and the amount currently adsorbed. It is coupled to the heat of the gas in the column.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file AdsorptionHeatAccumulation.h.

6.2.2 Function Documentation

6.2.2.1 template<> InputParameters validParams< AdsorptionHeatAccumulation > ( )

6.3 AdsorptionMassTransfer.h File Reference

Standard kernel for the transfer of mass via adsorption.

```
#include "Kernel.h"
```

Classes

- class AdsorptionMassTransfer

  AdsorptionMassTransfer class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< AdsorptionMassTransfer > ()

6.3.1 Detailed Description

Standard kernel for the transfer of mass via adsorption. This file creates a standard MOOSE kernel for the transfer of mass between the bulk gas of the fixed-bed and the adsorbent material in the column. The mass transfer is based on the amount of material in the bed and the solid adsorption variables.

Author

Austin Ladshaw

Date

01/29/2016

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2016, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file AdsorptionMassTransfer.h.

6.3.2 Function Documentation

6.3.2.1 template<> InputParameters validParams< AdsorptionMassTransfer >( )

6.4 BedHeatAccumulation.h File Reference

Time Derivative kernel for the accumulation of heat in a fixed-bed column.

```
#include "TimeDerivative.h"
```

Classes

- class BedHeatAccumulation

  BedHeatAccumulation class object inherits from TimeDerivative object.

Functions

- template<>
  InputParameters validParams< BedHeatAccumulation > ()

6.4.1 Detailed Description

Time Derivative kernel for the accumulation of heat in a fixed-bed column. This file creates a time derivative kernel to be used in the energy transport equations for adsorption in a fixed-bed column. It combines the retardation coefficient from a material property with the standard time derivative kernel object in MOOSE.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file BedHeatAccumulation.h.

6.4.2 Function Documentation

6.4.2.1 template<> InputParameters validParams< BedHeatAccumulation > ( )

6.5 BedMassAccumulation.h File Reference

Time Derivative kernel for the accumulation of mass of a species in a fixed-bed column.

```
#include "TimeDerivative.h"
```

Classes

- class BedMassAccumulation

  BedMassAccumulation class object inherits from TimeDerivative object.

Functions

- template<>
  InputParameters validParams< BedMassAccumulation > ()

6.5.1 Detailed Description

Time Derivative kernel for the accumulation of mass of a species in a fixed-bed column. This file creates a time derivative kernel to be used in the mass transport equations for adsorption in a fixed-bed column. It combines the retardation coefficient from a material property with the standard time derivative kernel object in MOOSE.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file BedMassAccumulation.h.

6.5.2 Function Documentation

6.5.2.1 template<> InputParameters validParams< BedMassAccumulation >( )

6.6 BedProperties.h File Reference

```
#include "Material.h"
```

Classes

- class BedProperties

  BedProperties class object inherits from Material object.

Functions

- template<>
  InputParameters validParams< BedProperties > ()

6.6.1 Function Documentation

6.6.1.1 template<> InputParameters validParams< BedProperties >( )

6.7 BedWallHeatTransfer.h File Reference

Standard kernel for the transfer of heat from the fixed-bed to the column wall.

```
#include "Kernel.h"
```

Classes

- class BedWallHeatTransfer

  BedWallHeatTransfer class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< BedWallHeatTransfer > ()

6.7.1 Detailed Description

Standard kernel for the transfer of heat from the fixed-bed to the column wall. This file creates a standard MOOSE kernel for the transfer of energy as heat between the bulk gas temperature of the fixed-bed and the temperature of the walls of the column. The heat transfer is based on the thickness of the wall and a bed-wall heat transfer coefficient. It is coupled to the heat of the gas in the column and is a primary kernel used in determining the heat of the wall.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file BedWallHeatTransfer.h.

6.7.2 Function Documentation

6.7.2.1 template<> InputParameters validParams< BedWallHeatTransfer > ( )

6.8 ColumnTemperatureIC.h File Reference

Initial Condition kernel for initial temperature in a fixed-bed column.

```
#include "InitialCondition.h"
```

Classes

- class ColumnTemperatureIC

  ColumnTemperatureIC class object inherits from InitialCondition object.

Functions

- template<>
  InputParameters validParams< ColumnTemperatureIC > ()

6.8.1 Detailed Description

Initial Condition kernel for initial temperature in a fixed-bed column. This file creates an initial condition for the temperature in the bed. The initial condition for temperature is assumed a constant value at all points in the bed. However, this can be modified later to include spatially varying initial conditions for temperature.

Note

If you want to have spatially varying initial conditions, you will need to modify the virtual value function of this kernel. Otherwise, it is assumed that the non-linear variable is initially constant at all points in the domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file ColumnTemperatureIC.h.

6.8.2 Function Documentation

6.8.2.1 template<> InputParameters validParams< ColumnTemperatureIC >( )

6.9 ConcentrationIC.h File Reference

Initial Condition kernel for initial concentration of a species in a fixed-bed column.

```
#include "InitialCondition.h"
```

Classes

- class ConcentrationIC

  ConcentrationIC class object inherits from InitialCondition object.

Functions

- template<>
  InputParameters validParams< ConcentrationIC > ()

6.9.1 Detailed Description

Initial Condition kernel for initial concentration of a species in a fixed-bed column. This file creates an initial condition for the concentration of a species in the bed. The initial condition for concentration is assumed a constant value at all points in the bed. However, this can be modified later to include varying initial conditions.

Note

If you want to have spatially varying initial conditions, you will need to modify the virtual value function of this kernel. Otherwise, it is assumed that the non-linear variable is initially constant at all points in the domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file ConcentrationIC.h.

6.9.2 Function Documentation

6.9.2.1 template<> InputParameters validParams< ConcentrationIC > ( )

6.10 CoupledLDF.h File Reference

Advanced kernel for a cross coupled linear driving force mechanism.

```
#include "LinearDrivingForce.h"
```

Classes

- class CoupledLDF

  CoupledLDF class object inherits from LinearDrivingForce object.

Macros

- #define COUPLEDLDF_H

Functions

- template<>
  InputParameters validParams< CoupledLDF > ()

6.10.1 Detailed Description

Advanced kernel for a cross coupled linear driving force mechanism. This file creates a standard MOOSE kernel for a coupled linear driving force type of mechanism that can be added to the non-linear residuals. It contains a boolean argument to determine whether the driving force is gaining or losing, a coefficient for the rate of the driving force, and a driving value to where the non-linear coupled variable is heading towards.

This file inherits from LinearDrivingForce.h

Author

Austin Ladshaw

Date

01/29/2016

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2016, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file CoupledLDF.h.

6.10.2 Macro Definition Documentation

6.10.2.1 #define COUPLEDLDF_H

Definition at line 43 of file CoupledLDF.h.

6.10.3 Function Documentation

6.10.3.1 template<> InputParameters validParams< CoupledLDF >( )

6.11 DGAdvection.h File Reference

Discontinous Galerkin kernel for advection.

```
#include "DGKernel.h"
#include <cmath>
```

Classes

- class DGAdvection

  DGAdvection class object inherits from DGKernel object.

Functions

- template<>
  InputParameters validParams< DGAdvection > ()

6.11.1 Detailed Description

Discontinous Galerkin kernel for advection. This file creates a discontinous Galerkin kernel for advection physics in a given domain. It is a generic advection kernel that is meant to be inherited from to make a more specific kernel for a given problem. The physical parameter in this kernel's formulation is a velocity vector. That vector can be built piecewise by the respective x, y, and z components of a velocity field at a given quadrature point.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGAdvection.h.

6.11.2 Function Documentation

6.11.2.1 template<> InputParameters validParams< DGAdvection > ( )

6.12 DGAnisotropicDiffusion.h File Reference

Discontinous Galerkin kernel for anisotropic diffusion.

```
#include "DGKernel.h"
#include <cmath>
```

Classes

- class DGAnisotropicDiffusion

  DGAnisotropicDiffusion class object inherits from DGKernel object.

Functions

- template<>
  InputParameters validParams< DGAnisotropicDiffusion > ()

6.12.1 Detailed Description

Discontinous Galerkin kernel for anisotropic diffusion. This file creates a discontinous Galerkin kernel for anisotropic diffusion in a given domain. It is a generic diffusion kernel that is meant to be inherited from to make a more specific kernel for a given problem. The physical parameter in this kernel's formulation is a diffusion tensor. That tensor can be built piecewise by the respective components of the tensor at a given quadrature point.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGAnisotropicDiffusion.h.

6.12.2 Function Documentation

6.12.2.1 template<> InputParameters validParams< DGAnisotropicDiffusion >( )

6.13 DGColumnHeatAdvection.h File Reference

Discontinous Galerkin kernel for energy advection in a fixed-bed column.

```
#include "DGAdvection.h"
```

Classes

- class DGColumnHeatAdvection

  DGColumnHeatAdvection class object inherits from DGAdvection object.

Macros

- #define DGCOLUMNHEATADVECTION_H

Functions

- template<>
  InputParameters validParams< DGColumnHeatAdvection > ()

6.13.1 Detailed Description

Discontinous Galerkin kernel for energy advection in a fixed-bed column. This file creates a discontinous Galerkin kernel for the advective heat transfer in a fixed-bed column. The advection portion of the energy transport equations involves the linear velocity in the system, as well as the gas density and gas heat capacity. Those parameters are given as material properties for the system.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnHeatAdvection.h.

6.13.2 Macro Definition Documentation

6.13.2.1 #define DGCOLUMNHEATADVECTION_H

Definition at line 47 of file DGColumnHeatAdvection.h.

6.13.3 Function Documentation

6.13.3.1 template<> InputParameters validParams< DGColumnHeatAdvection >( )

6.14 DGColumnHeatDispersion.h File Reference

Discontinous Galerkin kernel for energy dispersion in a fixed-bed column.

```
#include "DGAnisotropicDiffusion.h"
```

Classes

- class DGColumnHeatDispersion

  DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object.

Macros

- #define DGCOLUMNHEATDISPERSION_H

Functions

- template<>
  InputParameters validParams< DGColumnHeatDispersion > ()

6.14.1 Detailed Description

Discontinous Galerkin kernel for energy dispersion in a fixed-bed column. This file creates a discontinous Galerkin kernel for the dispersive heat transfer in a fixed-bed column. The dispersion portion of the energy transport equations involves the thermal conductivity in the system. That parameter is calculated in a material properties file and passed into this object for use the construction of residuals and Jacobians.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGColumnHeatDispersion.h.

6.14.2 Macro Definition Documentation

6.14.2.1 #define DGCOLUMNHEATDISPERSION_H

Definition at line 47 of file DGColumnHeatDispersion.h.

6.14.3 Function Documentation

6.14.3.1 template<> InputParameters validParams< DGColumnHeatDispersion > ( )

6.15 DGColumnMassAdvection.h File Reference

Discontinous Galerkin kernel for mass advection in a fixed-bed column.

```
#include "DGAdvection.h"
```

Classes

- class DGColumnMassAdvection

  DGColumnMassAdvection class object inherits from DGAdvection object.

Functions

- template<>
  InputParameters validParams< DGColumnMassAdvection > ()

6.15.1 Detailed Description

Discontinous Galerkin kernel for mass advection in a fixed-bed column. This file creates a discontinous Galerkin kernel for the advective mass transfer in a fixed-bed column. The advection portion of the mass transport equations involves the linear velocity in the system. That parameter is given as material property.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGColumnMassAdvection.h.

6.15.2 Function Documentation

6.15.2.1 template<> InputParameters validParams< DGColumnMassAdvection > ( )

6.16 DGColumnMassDispersion.h File Reference

Discontinous Galerkin kernel for mass dispersion in a fixed-bed column.

```
#include "DGAnisotropicDiffusion.h"
```

Classes

- class DGColumnMassDispersion

  DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object.

Functions

- template<>
  InputParameters validParams< DGColumnMassDispersion > ()

6.16.1 Detailed Description

Discontinous Galerkin kernel for mass dispersion in a fixed-bed column. This file creates a discontinous Galerkin kernel for the dispersive mass transfer in a fixed-bed column. The dispersion portion of the mass transport equations involves the molecular diffusivity of a species in the system, as well as the overall axial dispersion of material caused by mechanical mixing and molecular diffusion. Those parameters are calculated in a material properties file and passed into this object for use the construction of residuals and Jacobians.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGColumnMassDispersion.h.

6.16.2 Function Documentation

6.16.2.1 template<> InputParameters validParams< DGColumnMassDispersion > ( )

6.17 DGColumnWallHeatFluxBC.h File Reference

Boundary Condition kernel for the heat flux across the wall of the fixed-bed column.

```
#include "DGFluxBC.h"
```

Classes

- class DGColumnWallHeatFluxBC

  DGColumnWallHeatFluxBC class object inherits from DGFluxBC object.

Functions

- template<>
  InputParameters validParams< DGColumnWallHeatFluxBC > ()

6.17.1 Detailed Description

Boundary Condition kernel for the heat flux across the wall of the fixed-bed column. This file creates a boundary condition kernel for the heat flux across the boundary of the walls of the column in the fixed-bed adsorber. It inherits from the DGFluxBC, which acts as a generic flux BC module. This kernel is coupled to the wall temperature variable, as well as the material properties for thermal conductivity and the bed-wall heat transfer coefficient.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGColumnWallHeatFluxBC.h.

6.17.2 Function Documentation

6.17.2.1 template<> InputParameters validParams< DGColumnWallHeatFluxBC > ( )

6.18 DGColumnWallHeatFluxLimitedBC.h File Reference

Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall.

```
#include "DGFluxLimitedBC.h"
```

Classes

- class DGColumnWallHeatFluxLimitedBC

  DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

Functions

- template<>
  InputParameters validParams< DGColumnWallHeatFluxLimitedBC > ()

6.18.1 Detailed Description

Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall. This file creates a boundary condition that mimics a Dirichlet boundary condition for the heat of the column at the wall. A true Dirichlet boundary condition does not exist in DG methods. However, this will create a weak form that imposes a constraint that the solution must be of a certain value at the boundary.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGColumnWallHeatFluxLimitedBC.h.

6.18.2 Function Documentation

6.18.2.1 template<> InputParameters validParams< DGColumnWallHeatFluxLimitedBC > ( )

6.19 DGFluxBC.h File Reference

Boundary Condition kernel for the flux across a boundary of the domain.

```
#include "IntegratedBC.h"
#include "libmesh/vector_value.h"
```

Classes

- class DGFluxBC

  DGFluxBC class object inherits from IntegratedBC object.

Functions

- template<>
  InputParameters validParams< DGFluxBC > ()

6.19.1 Detailed Description

Boundary Condition kernel for the flux across a boundary of the domain. This file creates a generic boundary condition kernel for the flux of material accross a boundary. The flux is based on a diffusivity tensor and a velocity vector and is valid in all directions and all boundaries of a DG method. Since the DG method's flux boundary conditions are essitially the same for input and ouput boundaries, this kernel will check the sign of the flux normal to the boundary and determine automattically whether it is an output or input boundary, then apply the appropriate conditions.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGFluxBC.h.

6.19.2 Function Documentation

6.19.2.1 template<> InputParameters validParams< DGFluxBC > ( )

6.20 DGFluxLimitedBC.h File Reference

Boundary Condition kernel to mimic a Dirichlet BC for DG methods.

```
#include "IntegratedBC.h"
#include "libmesh/vector_value.h"
```

Classes

- class DGFluxLimitedBC

  DGFluxLimitedBC class object inherits from IntegratedBC object.

Functions

- template<>
  InputParameters validParams< DGFluxLimitedBC > ()

6.20.1 Detailed Description

Boundary Condition kernel to mimic a Dirichlet BC for DG methods. This file creates a boundary condition kernel to impose a dirichlet-like boundary condition in DG methods. True DG methods do not have Dirichlet boundary conditions, so this kernel seeks to impose a constraint on the inlet of a boundary that is met if the value of a variable at the inlet boundary is equal to the finite element solution at that boundary. When the condition is not met, the residuals get penalyzed until the condition is met.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGFluxLimitedBC.h.

6.20.2 Function Documentation

6.20.2.1 template<> InputParameters validParams< DGFluxLimitedBC > ( )

6.21 DGHeatFluxBC.h File Reference

Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column.

```
#include "DGFluxBC.h"
```

Classes

- class DGHeatFluxBC

  DGHeatFluxBC class object inherits from DGFluxBC object.

Functions

- template<>
  InputParameters validParams< DGHeatFluxBC > ()

6.21.1 Detailed Description

Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column. This file creates a boundary condition kernel for the heat flux across the boundary of the ends of the column in the fixed-bed adsorber. It inherits from the DGFluxBC, which acts as a generic flux BC module. This kernel is coupled to the column temperature variable, as well as the material properties for thermal conductivity, gas density and heat capacity, and the velocity in the domain.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGHeatFluxBC.h.

6.21.2 Function Documentation

6.21.2.1 template<> InputParameters validParams< DGHeatFluxBC > ( )

6.22 DGHeatFluxLimitedBC.h File Reference

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet.

```
#include "DGFluxLimitedBC.h"
```

Classes

- class DGHeatFluxLimitedBC
  DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

Functions

- template<>
  InputParameters validParams< DGHeatFluxLimitedBC > ()

6.22.1 Detailed Description

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet. This file creates a dirichlet-like boundary condition kernel for the column temperature at the inlet of the system. The outlet boundary condition would remain unchanged from the standard DG form of the boundaries. Only the inlet BC is affected by this file. See FluxLimitedBC.h for more details.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGHeatFluxLimitedBC.h.

6.22.2 Function Documentation

6.22.2.1 template<> InputParameters validParams< DGHeatFluxLimitedBC >( )

6.23 DGMassFluxBC.h File Reference

Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column.

```
#include "DGFluxBC.h"
```

Classes

- class DGMassFluxBC

  DGMassFluxBC class object inherits from DGFluxBC object.

Functions

- template<>
  InputParameters validParams< DGMassFluxBC > ()

6.23.1 Detailed Description

Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column. This file creates a boundary condition kernel for the mass flux across the boundary of the ends of the column in the fixed-bed adsorber. It inherits from the DGFluxBC, which acts as a generic flux BC module. This kernel is coupled to the column temperature variable, as well as the material properties for thermal conductivity, gas density and heat capacity, and the velocity in the domain.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGMassFluxBC.h.

6.23.2 Function Documentation

6.23.2.1 template<> InputParameters validParams< DGMassFluxBC > ( )

6.24 DGMassFluxLimitedBC.h File Reference

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet.

```
#include "DGFluxLimitedBC.h"
```

Classes

- class DGMassFluxLimitedBC

  DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object.

Functions

- template<>
  InputParameters validParams< DGMassFluxLimitedBC > ()

6.24.1 Detailed Description

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet. This file creates a dirichlet-like boundary condition kernel for the gas species concentration at the inlet of the system. The outlet boundary condition would remain unchanged from the standard DG form of the boundaries. Only the inlet BC is affected by this file. See FluxLimitedBC.h for more details.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGMassFluxLimitedBC.h.

6.24.2 Function Documentation

6.24.2.1 template<> InputParameters validParams< DGMassFluxLimitedBC > ( )

6.25 DgospreyApp.h File Reference

Registration object for creating a registering DGOSPREY kernels.

```
#include "MooseApp.h"
```

Classes

- class DgospreyApp
  DgospreyApp inherits from the MooseApp object.

Functions

- template<>
  InputParameters validParams< DgospreyApp > ()

6.25.1 Detailed Description

Registration object for creating a registering DGOSPREY kernels. This file is responsible for registering all DGOSPREY kernels in the MOOSE framework. Any additional kernel developed under DGOSPREY must be included and registered in this structure. This structure is required by MOOSE in order to have the DGOSPREY objects interact with the underlying MOOSE solvers and finite element framework.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

> This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DgospreyApp.h.

6.25.2 Function Documentation

6.25.2.1 template<> InputParameters validParams< DgospreyApp > ( )

6.26 DgospreyRevision.h File Reference

Macros

- #define DGOSPREY_REVISION "git commit 9890b6c on 2014-09-24"

6.26.1 Macro Definition Documentation

6.26.1.1 #define DGOSPREY_REVISION "git commit 9890b6c on 2014-09-24"

Definition at line 6 of file base/DgospreyRevision.h.

6.27 DgospreyRevision.h File Reference

Macros

- #define DGOSPREY_REVISION "git commit f1c4b0d on 2016-01-12"

6.27.1 Macro Definition Documentation

6.27.1.1 #define DGOSPREY_REVISION "git commit f1c4b0d on 2016-01-12"

Definition at line 6 of file DgospreyRevision.h.

6.28 egret.h File Reference

Estimation of Gas-phase pRopErTies.

```
#include "macaw.h"
```

Classes

- struct PURE_GAS

  Data structure holding all the parameters for each pure gas spieces.
- struct MIXED_GAS

  Data structure holding information necessary for computing mixed gas properties.

Macros

- #define EGRET_HPP_
- #define Rstd 8.3144621

  Gas Constant in J/K/mol (or) L∗kPa/K/mol (Standard Units)
- #define RE3 8.3144621E+3

  Gas Constant in cm^3∗kPa/K/mol (Convenient for density calculations)
- #define Po 100.0

  Standard state pressure (kPa)
- #define Cstd(p, T) ((p)/(Rstd∗T))

  Calculation of concentration/density from partial pressure (Cstd = mol/L)
- #define CE3(p, T) ((p)/(RE3∗T))

  Calculation of concentration/density from partial pressure (CE3 = mol/cm^3)
- #define Pstd(c, T) ((c)∗Rstd∗T)

  Calculation of partial pressure from concentration/density (c = mol/L)
- #define PE3(c, T) ((c)∗RE3∗T)

  Calculation of partial pressure from concentration/density (c = mol/cm^3)
- #define Nu(mu, rho) ((mu)/(rho))

  Calculation of kinematic viscosity from dynamic viscosity and density (cm^2/s)
- #define PSI(T) (0.873143 + (0.000072375∗T))

  Calculation of temperature correction factor for dynamic viscosity.
- #define Dp_ij(Dij, PT) ((PT∗Dij)/Po)

  Calculation of the corrected binary diffusivity (cm^2/s)
- #define D_ij(MWi, MWj, rhoi, rhoj, mui, muj) ((4.0 / sqrt(2.0)) ∗ pow(((1/MWi)+(1/MWj)),0.5)) / pow((pow((pow((rhoi/(1.385∗mui)),2.0)/MWi),0.25)+ pow((pow((rhoj/(1.385∗muj)),2.0)/MWj),0.25)),2.0)

  Calculation of binary diffusion based on MW, density, and viscosity info (cm^2/s)
- #define Mu(muo, To, C, T) (muo ∗ ((To + C)/(T + C)) ∗ pow((T/To),1.5))

  Calculation of single species viscosity from Sutherland's Equ. (g/cm/s)
- #define D_ii(rhoi, mui) (1.385∗mui/rhoi)

  Calculation of self-diffusivity (cm^2/s)
- #define ReNum(u, L, nu) (u∗L/nu)

  Calculation of the Reynold's Number (-)
- #define ScNum(nu, D) (nu/D)

  Calculation of the Schmidt Number (-)
- #define FilmMTCoeff(D, L, Re, Sc) ((D/L)∗(2.0 + (1.1∗pow(Re,0.6)∗pow(Sc,0.3))))

  Calculation of film mass transfer coefficient (cm/s)

Functions

- int initialize_data (int N, MIXED_GAS ∗gas_dat)

  Function to initialize the MIXED_GAS structure based on number of gas species.
- int set_variables (double PT, double T, double us, double L, std::vector< double > &y, MIXED_GAS ∗gas_dat)

  Function to set the values of the parameters in the gas phase.
- int calculate_properties (MIXED_GAS ∗gas_dat)

  Function to calculate the gas properties based on information in MIXED_GAS.

6.28.1 Detailed Description

Estimation of Gas-phase pRopErTies. egret.cpp

This file is responsible for estimating various temperature, pressure, and concentration dependent parameters to be used in other models for gas phase adsorption, mass transfer, and or mass transport. The goal of this file is to eliminate redundancies in code such that the higher level programs operate more efficiently and cleanly. Calculations made here are based on kinetic theory of gases, ideal gas law, and some emperical models that were developed to account for changes in density and viscosity with changes in temperature between standard temperatures and up to 1000 K.

Author

Austin Ladshaw

Date

01/29/2015

Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file egret.h.

6.28.2 Macro Definition Documentation

6.28.2.1 #define CE3(*p, T*) ((p)/(RE3∗T))

Calculation of concentration/density from partial pressure (CE3 = mol/cm^3)

Definition at line 42 of file egret.h.

6.28.2.2 #define Cstd(*p, T*) ((p)/(Rstd∗T))

Calculation of concentration/density from partial pressure (Cstd = mol/L)

Definition at line 38 of file egret.h.

6.28.2.3 #define D_ii(*rhoi, mui*) (1.385∗mui/rhoi)

Calculation of self-diffusivity (cm^2/s)

Definition at line 74 of file egret.h.

6.28.2.4 #define D_ij(*MWi, MWj, rhoi, rhoj, mui, muj*) ((4.0 / sqrt(2.0)) ∗ pow(((1/MWi)+(1/MWj)),0.5)) / pow((pow((pow((rhoi/(1.385∗mui)),2.0)/MWi),0.25)+ pow((pow((rhoj/(1.385∗muj)),2.0)/MWj),0.25)),2.0)

Calculation of binary diffusion based on MW, density, and viscosity info (cm^2/s)

Definition at line 66 of file egret.h.

6.28.2.5 #define Dp_ij(*Dij, PT*) ((PT∗Dij)/Po)

Calculation of the corrected binary diffusivity (cm^2/s)

Definition at line 62 of file egret.h.

6.28.2.6 #define EGRET_HPP_

Definition at line 23 of file egret.h.

6.28.2.7 #define FilmMTCoeff(*D, L, Re, Sc*) ((D/L)∗(2.0 + (1.1∗pow(Re,0.6)∗pow(Sc,0.3))))

Calculation of film mass transfer coefficient (cm/s)

Definition at line 86 of file egret.h.

6.28.2.8 #define Mu(*muo, To, C, T*) (muo ∗ ((To + C)/(T + C)) ∗ pow((T/To),1.5))

Calculation of single species viscosity from Sutherland's Equ. (g/cm/s)

Definition at line 70 of file egret.h.

6.28.2.9 #define Nu(*mu, rho*) ((mu)/(rho))

Calculation of kinematic viscosity from dynamic viscosity and density (cm^2/s)

Definition at line 54 of file egret.h.

6.28.2.10 #define PE3(*c, T*) ((c)∗RE3∗T)

Calculation of partial pressure from concentration/density (c = mol/cm^3)

Definition at line 50 of file egret.h.

6.28.2.11 #define Po 100.0

Standard state pressure (kPa)

Definition at line 34 of file egret.h.

6.28.2.12 #define PSl(*T*) (0.873143 + (0.000072375∗T))

Calculation of temperature correction factor for dynamic viscosity.

Definition at line 58 of file egret.h.

6.28.2.13 #define Pstd(*c, T*) ((c)∗Rstd∗T)

Calculation of partial pressure from concentration/density (c = mol/L)

Definition at line 46 of file egret.h.

6.28.2.14 #define RE3 8.3144621E+3

Gas Constant in cm^3∗kPa/K/mol (Convenient for density calculations)

Definition at line 30 of file egret.h.

6.28.2.15 #define ReNum(*u, L, nu*) (u∗L/nu)

Calculation of the Reynold's Number (-)

Definition at line 78 of file egret.h.

6.28.2.16 #define Rstd 8.3144621

Gas Constant in J/K/mol (or) L∗kPa/K/mol (Standard Units)

Definition at line 26 of file egret.h.

6.28.2.17 #define ScNum(*nu, D*) (nu/D)

Calculation of the Schmidt Number (-)

Definition at line 82 of file egret.h.

6.28.3 Function Documentation

6.28.3.1 int calculate_properties (MIXED_GAS * gas_dat)

Function to calculate the gas properties based on information in MIXED_GAS.

This function uses the kinetic theory of gases, combined with other semi-empirical models, to predict and approximate several properties of the mixed gas phase that might be necessary when running any gas dynamical simulation. This includes mass and energy transfer equations, as well as adsorption kinetics in porous adsorbents.

6.28.3.2 int initialize_data (int N, MIXED_GAS * gas_dat)

Function to initialize the MIXED_GAS structure based on number of gas species.

This function will initialize the sizes of all vector objects in the MIXED_GAS structure based on the number of gas species indicated by N.

6.28.3.3 int set_variables (double PT, double T, double us, double L, std::vector< double > & y, MIXED_GAS * gas_dat)

Function to set the values of the parameters in the gas phase.

The gas phase properties are a function of total pressure, gas temperature, gas velocity, characteristic length, and the mole fractions of each species in the gas phase. Prior to calculating the gas phase properties, these parameters must be set and updated as they change.

Parameters

PT	total gas pressure in kPa
T	gas temperature in K
us	gas velocity in cm/s
L	characteristic length in cm (this depends on the particular system)
y	vector of gas mole fractions of each species in the mixture
gas_dat	pointer to the MIXED_GAS data structure

6.29 error.h File Reference

All error types are defined here.

```
#include <iostream>
```

Macros

- #define mError(i)

Enumerations

- enum error_type {
  generic_error, file_dne, indexing_error, magpie_reverse_error,
  simulation_fail, invalid_components, invalid_boolean, invalid_molefraction,
  invalid_gas_sum, invalid_solid_sum, scenario_fail, out_of_bounds,
  non_square_matrix, dim_mis_match, empty_matrix, opt_no_support,
  invalid_fraction, ortho_check_fail, unstable_matrix, no_diffusion,
  negative_mass, negative_time, matvec_mis_match, arg_matrix_same,
  singular_matrix, matrix_too_small, invalid_size, nullptr_func,
  invalid_norm, vector_out_of_bounds, zero_vector, tensor_out_of_bounds,
  non_real_edge, nullptr_error, invalid_atom, invalid_proton,
  invalid_neutron, invalid_electron, invalid_valence, string_parse_error,
  unregistered_name, rxn_rate_error, invalid_species, duplicate_variable,
  missing_information, invalid_type, key_not_found, anchor_alias_dne,
  initial_error, not_a_token, read_error, invalid_console_input }

  List of names for error type.

Functions

- void error (int flag)

  Error function customizes output message based on flag.

6.29.1 Detailed Description

All error types are defined here. error.cpp

This file defines all the different errors that may occur in any simulation in any file. Those errors are recognized by an enum with is then passed through to the error.cpp file that customizes the error message to the console. A macro will also print out the file name and line number where the error occured.

Author

Austin Ladshaw

Date

04/28/2014

Copyright

Definition in file error.h.

6.29.2 Macro Definition Documentation

6.29.2.1 #define mError(*i*)

Value:

```
{error(i);          \
std::cout << "Source: " << __FILE__ << "\nLine: " << __LINE__ << std::endl;}
```

Definition at line 22 of file error.h.

Referenced by Matrix< T >::adjoint(), Matrix< T >::cofactor(), Matrix< T >::columnExtend(), Matrix< T >::columnExtract(), Matrix< T >::columnProjection(), Matrix< T >::columnReplace(), Matrix< T >::columnVectorFill(), Matrix< T >::ConstantICFill(), Matrix< T >::determinate(), Matrix< T >::diagonalSolve(), Matrix< T >::dirichletBCFill(), Matrix< T >::Display(), Matrix< T >::edit(), Matrix< T >::inner_product(), Matrix< T >::IntegralAvg(), Matrix< T >::IntegralTotal(), Matrix< T >::inverse(), Matrix< T >::ladshawSolve(), Matrix< T >::lowerHessenberg2Triangular(), Matrix< T >::lowerHessenbergSolve(), Matrix< T >::lowerTriangularSolve(), Matrix< T >::operator()(), Matrix< T >::operator*(), Matrix< T >::operator+(), Matrix< T >::operator-(), Matrix< T >::rowExtend(), Matrix< T >::rowExtract(), Matrix< T >::rowReplace(), Matrix< T >::set_size(), Matrix< T >::SolnTransform(), Matrix< T >::sphericalAvg(), Matrix< T >::sphericalBCFill(), Matrix< T >::transpose(), Matrix< T >::transpose_multiply(), Matrix< T >::tridiagonalFill(), Matrix< T >::tridiagonalSolve(), Matrix< T >::tridiagonalVectorFill(), Matrix< T >::upperHessenberg2Triangular(), Matrix< T >::upperHessenbergSolve(), and Matrix< T >::upperTriangularSolve().

6.29.3 Enumeration Type Documentation

6.29.3.1 enum error_type

List of names for error type.

Enumerator

generic_error

file_dne

indexing_error

magpie_reverse_error

simulation_fail

invalid_components

invalid_boolean

invalid_molefraction

invalid_gas_sum

invalid_solid_sum

scenario_fail

out_of_bounds

non_square_matrix

dim_mis_match

empty_matrix

opt_no_support

invalid_fraction

ortho_check_fail

unstable_matrix

no_diffusion

negative_mass

negative_time

matvec_mis_match

arg_matrix_same

singular_matrix

matrix_too_small

invalid_size

nullptr_func
invalid_norm
vector_out_of_bounds
zero_vector
tensor_out_of_bounds
non_real_edge
nullptr_error
invalid_atom
invalid_proton
invalid_neutron
invalid_electron
invalid_valence
string_parse_error
unregistered_name
rxn_rate_error
invalid_species
duplicate_variable
missing_information
invalid_type
key_not_found
anchor_alias_dne
initial_error
not_a_token
read_error
invalid_console_input

Definition at line 28 of file error.h.

6.29.4 Function Documentation

6.29.4.1 void error (int *flag*)

Error function customizes output message based on flag.

This error function is reference in the error.cpp file, but is not called by any other file. Instead, all other files call the mError(i) macro that expands into this error function call plus prints out the file name and line number where the error occured.

6.30 finch.h File Reference

Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme.

```
#include "macaw.h"
#include "lark.h"
```

Classes

- struct FINCH_DATA

  Data structure for the FINCH object.

Enumerations

- enum finch_solve_type { FINCH_Picard, LARK_Picard, LARK_PJFNK }

  List of enum options to define the solver type in FINCH.
- enum finch_coord_type { Cartesian, Cylindrical, Spherical }

  List of enum options to define the coordinate system in FINCH.

Functions

- double max (std::vector< double > &values)

  Function returns the maximum in a list of values.
- double min (std::vector< double > &values)

  Function returns the minimum in a list of values.
- double minmod (std::vector< double > &values)

  Function returns the result of the minmod function acting on a list of values.
- int uTotal (FINCH_DATA *dat)

  Function integrates the conserved quantity to return it's total in the domain.
- int uAverage (FINCH_DATA *dat)

  Function integrates the conserved quantity to reture it's average in the domain.
- int check_Mass (FINCH_DATA *dat)

  Function checks the unp1 vector for negative values and will adjust if needed.
- int l_direct (FINCH_DATA *dat)

  Function solves the discretized FINCH problem directly by assuming it is linear.
- int lark_picard_step (const Matrix< double > &x, Matrix< double > &G, const void *data)

  Function to perform the necessary LARK Picard iterative method (not typically used)
- int nl_picard (FINCH_DATA *dat)

  Function to solve the discretized FINCH problem iteratively by assuming it is non-linear.
- int setup_FINCH_DATA (int(*user_callroutine)(const void *user_data), int(*user_setic)(const void *user_data), int(*user_timestep)(const void *user_data), int(*user_preprocess)(const void *user_data), int(*user_solve)(const void *user_data), int(*user_setparams)(const void *user_data), int(*user_discretize)(const void *user_data), int(*user_bcs)(const void *user_data), int(*user_res)(const Matrix< double > &x, Matrix< double > &res, const void *user_data), int(*user_precon)(const Matrix< double > &b, Matrix< double > &p, const void *user_data), int(*user_postprocess)(const void *user_data), int(*user_reset)(const void *user_data), FINCH_DATA *dat, const void *param_data)

  Function to setup memory and set user defined functions into the FINCH object.
- void print2file_dim_header (FILE *Output, FINCH_DATA *dat)

  Function will print out a dimension header for FINCH output.
- void print2file_time_header (FILE *Output, FINCH_DATA *dat)

  Function will print out a time header for FINCH output.
- void print2file_result_old (FILE *Output, FINCH_DATA *dat)

  Function will print out the old results to the variable u.
- void print2file_result_new (FILE *Output, FINCH_DATA *dat)

  Function will print out the new results to the variable u.
- void print2file_newline (FILE *Output, FINCH_DATA *dat)

  Function will force print out a blank line.
- void print2file_tab (FILE *Output, FINCH_DATA *dat)

  Function will force print out a tab.
- int default_execution (const void *user_data)

  Default executioner function for FINCH.
- int default_ic (const void *user_data)

  Default initial conditions function for FINCH.
- int default_timestep (const void *user_data)

Default time step function for FINCH.

- int default_preprocess (const void *user_data)

  Default preprocesses function for FINCH.
- int default_solve (const void *user_data)

  Default solve function for FINCH.
- int default_params (const void *user_data)

  Default params function for FINCH.
- int minmod_discretization (const void *user_data)

  Minmod Discretization function for FINCH.
- int vanAlbada_discretization (const void *user_data)

  Van Albada Discretization function for FINCH.
- int ospre_discretization (const void *user_data)

  Ospre Discretization function for FINCH.
- int default_bcs (const void *user_data)

  Default boundary conditions function for FINCH.
- int default_res (const Matrix< double > &x, Matrix< double > &res, const void *user_data)

  Default residual function for FINCH.
- int default_precon (const Matrix< double > &b, Matrix< double > &p, const void *user_data)

  Default preconditioning function for FINCH.
- int default_postprocess (const void *user_data)
- int default_reset (const void *user_data)

  Default reset function for FINCH.

6.30.1 Detailed Description

Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme. finch.cpp

This is a conservative finite differences scheme based on the Kurganov and Tadmoor (2000) MUSCL scheme for non-linear conservation laws. It can solve 1-D conservation law problems in three different coordinate systems: (i) Cartesian - axial, (ii) Cylindrical - radial, and (iii) Spherical - radial. It is the backbone algorithm behind all 1-D PDE problems in the ecosystem software.

The form of the general conservation law problem that FINCH solves is...

z^d*R*du/dt = d/dz(z^d*D*du/dz) - d/dz(z^d*v*u) - z^d*k*u + z^d*S

where R, D, v, k, and S are the parameters of the problem and d, z, and u are the coordinates, spatial dimension, and conserved quantities, respectively. The parameter R is a retardation coefficient, D is a diffusion coefficient, v is a velocity, k is a reaction coefficient, and S is a forcing function or source/sink term.

FINCH supports the use of both Dirichlet and Neuman boundary conditions as the input/inlet condition and uses the No Flux (or Natural) boundary condition for the output/outlet of the domain. For radial problems, the outlet is always taken to the the center of the cylindrical or spherical particle. This enforces the symmetry of the problem. For axial problems, the outlet is determined by the sign of the velocity term and is therefore choosen by the routine based on the actual flow direction in the domain.

Parameters of the problem can be coupled to the variable u and also be functions of space and time. The coupling of the parameters with the variable forces the problem to become non-linear, which requires iteration to solve. The default iterative method is a built-in Picard's method. This method is equivalent to an inexact Newton method, because we use the Linear Solve of this system as a weak approximation to the non-linear solve. Generally, this method is sufficient and is the most efficient. However, if a problem is particularly difficult to solve, then we can call some of the non-linear solvers developed in LARK. If PJFNK is used, then the Linear Solve for the FINCH problem is used as the Preconditioner for the Linear Solve in PJFNK.

This algorithm comes packaged with three different slope limiter functions to stabilize the velocity term for highly advectively dominate problems. The available slope limiters are: (i) minmod, (ii) van Albada, and (iii) ospre. By default, the FINCH setup function will set the slope limiter to ospre, because this method provides a reasonable compromise between accuracy and efficiency.

Slope Limiter Stats:

minmod -> Highest Accuracy, Lowest Efficiency

van Albada -> Lowest Accuracy, Highest Efficiency

ospre -> Average Accuracy, Average Efficiency

Author

Austin Ladshaw

Date

01/29/2015

Copyright

Definition in file finch.h.

6.30.2 Enumeration Type Documentation

6.30.2.1 enum finch_coord_type

List of enum options to define the coordinate system in FINCH.

Enumerator

Cartesian

Cylindrical

Spherical

Definition at line 65 of file finch.h.

6.30.2.2 enum finch_solve_type

List of enum options to define the solver type in FINCH.

Enumerator

FINCH_Picard

LARK_Picard

LARK_PJFNK

Definition at line 62 of file finch.h.

6.30.3 Function Documentation

6.30.3.1 int check_Mass (FINCH_DATA * *dat*)

Function checks the unp1 vector for negative values and will adjust if needed.

This function can be turned off or on in the FINCH_DATA structure. Typically, you will want to leave this on so that the routine does not return negative values for u. However, if you want to get negative values of u, then turn this option off.

6.30.3.2 int default_bcs (const void * *user_data*)

Default boundary conditions function for FINCH.

The default boundary conditions function for FINCH assumes the user_data parameter is the FINCH_DATA structure and sets the boundary conditions according to the type of problem requested. The input BCs will always be either Neumann or Dirichlet and the output BC will always be a zero flux Neumann BC.

6.30.3.3 int default_execution (const void * *user_data*)

Default executioner function for FINCH.

The default executioner function for FINCH assumes the user_data parameter is the FINCH_DATA structure and calls the preprocesses, solve, postprocesses, checkMass, uTotal, and uAverage functions in that order.

6.30.3.4 int default_ic (const void * *user_data*)

Default initial conditions function for FINCH.

The default initial condition function for FINCH assumes the user_data parameter is the FINCH_DATA structure and sets the initial values of all system parameters according to the given constants in that structure.

6.30.3.5 int default_params (const void * *user_data*)

Default params function for FINCH.

The default params function for FINCH assumes the user_data parameter is the FINCH_DATA structure and sets the values of all parameters at all nodes equal to the values of those parameters at the boundaries.

6.30.3.6 int default_postprocess (const void * *user_data*)

The default postprocesses function for FINCH assumes the user_data parameter is the FINCH_DATA structure and does nothing.

6.30.3.7 int default_precon (const Matrix< double > & *b*, Matrix< double > & *p*, const void * *user_data*)

Default preconditioning function for FINCH.

The default preconditioning function for FINCH assumes the user_data parameter is the FINCH_DATA structure and performs a tridiagonal linear solve using a Modified Thomas Algorithm. This preconditioner will solve the linear problem exactly if there is no advective portion of the physics. Additionally, this preconditioner is also used as the basis for forming the default FINCH non-linear iterations and is sufficient for solving most problems.

6.30.3.8 int default_preprocess (const void * *user_data*)

Default preprocesses function for FINCH.

The default preprocesses function for FINCH assumes the user_data parameter is the FINCH_DATA structure and does nothing.

6.30.3.9 int default_res (const Matrix< double > & *x*, Matrix< double > & *res*, const void * *user_data*)

Default residual function for FINCH.

The default residual function for FINCH assumes the user_data parameter is the FINCH_DATA structure and calls the setparams function (passing the param_data structure), the discretization function, and the set BCs functions, in that order. It then forms the implicit and explicit side residuals that go into the iterative solver.

6.30.3.10 int default_reset (const void * *user_data*)

Default reset function for FINCH.

The default reset function for FINCH assumes the user_data parameter is the FINCH_DATA structure and sets all old state parameters and variables to the new state.

6.30.3.11 int default_solve (const void * *user_data*)

Default solve function for FINCH.

The default solve function for FINCH assumes the user_data parameter is the FINCH_DATA structure and calls the corresponding solution method depending on the users conditions.

6.30.3.12 int default_timestep (const void * *user_data*)

Default time step function for FINCH.

The default time step function for FINCH assumes the user_data parameter is the FINCH_DATA structure and sets the time step to 1/2 the mesh size or bases the time step off of the CFL condition if the problem is not being solved iteratively and involves an advective portion.

6.30.3.13 int l_direct (FINCH_DATA * *dat*)

Function solves the discretized FINCH problem directly by assuming it is linear.

6.30.3.14 int lark_picard_step (const Matrix< double > & *x,* Matrix< double > & *G,* const void * *data*)

Function to perform the necessary LARK Picard iterative method (not typically used)

6.30.3.15 double max (std::vector< double > & *values*)

Function returns the maximum in a list of values.

6.30.3.16 double min (std::vector< double > & *values*)

Function returns the minimum in a list of values.

6.30.3.17 double minmod (std::vector< double > & *values*)

Function returns the result of the minmod function acting on a list of values.

6.30.3.18 int minmod_discretization (const void * *user_data*)

Minmod Discretization function for FINCH.

The minmod discretization function for FINCH assumes the user_data parameter is the FINCH_DATA structure and discretizes the time and space portion of the problem with 2nd order finite differences and uses the minmod slope limiter function to stabilize the advective physics.

6.30.3.19 int nl_picard (FINCH_DATA * *dat*)

Function to solve the discretized FINCH problem iteratively by assuming it is non-linear.

Note

If the problem is actually linear, then this will solve it in one iteration. So it may be best to always assume the problem is non-linear.

6.30.3.20 int ospre_discretization (const void * *user_data*)

Ospre Discretization function for FINCH.

The ospre discretization function for FINCH assumes the user_data parameter is the FINCH_DATA structure and discretizes the time and space portion of the problem with 2nd order finite differences and uses the ospre slope limiter function to stabilize the advective physics. This is the default discretization function.

6.30.3.21 void print2file_dim_header (FILE * *Output,* FINCH_DATA * *dat*)

Function will print out a dimension header for FINCH output.

6.30.3.22 void print2file_newline (FILE * *Output,* FINCH_DATA * *dat*)

Function will force print out a blank line.

6.30.3.23 void print2file_result_new (FILE * *Output,* FINCH_DATA * *dat*)

Function will print out the new results to the variable u.

6.30.3.24 void print2file_result_old (FILE * *Output,* FINCH_DATA * *dat*)

Function will print out the old results to the variable u.

6.30.3.25 void print2file_tab (FILE * *Output,* FINCH_DATA * *dat*)

Function will force print out a tab.

6.30.3.26 void print2file_time_header (FILE * *Output,* FINCH_DATA * *dat*)

Function will print out a time header for FINCH output.

6.30.3.27 int setup_FINCH_DATA (int(*)(const void *user_data) *user_callroutine,* int(*)(const void *user_data) *user_setic,* int(*)(const void *user_data) *user_timestep,* int(*)(const void *user_data) *user_preprocess,* int(*)(const void *user_data) *user_solve,* int(*)(const void *user_data) *user_setparams,* int(*)(const void *user_data) *user_discretize,* int(*)(const void *user_data) *user_bcs,* int(*)(const Matrix< double > &x, Matrix< double > &res, const void *user_data) *user_res,* int(*)(const Matrix< double > &b, Matrix< double > &p, const void *user_data) *user_precon,* int(*)(const void *user_data) *user_postprocess,* int(*)(const void *user_data) *user_reset,* FINCH_DATA * *dat,* const void * *param_data*)

Function to setup memory and set user defined functions into the FINCH object.

This function MUST be called prior to running any FINCH based simulation. However, you are only every required to provide this function with the FINCH_DATA pointer. It is recommended, however, that you do provide the user_-setparams and param_data pointers, as these will likely vary significantly from problem to problem.

After the problem is setup in memory, you do not technically have to have FINCH call all of it's own functions. You can write your own executioner, initial conditions, and other functions and decided how and when everything is called. Then just call the solve function in FINCH_DATA when you want to use the FINCH solver. This is how FINCH is used in SKUA, SCOPSOWL, DOGFISH, and MONKFISH.

Parameters

user_callroutine	function pointer the the call routine function
user_setic	function pointer to set initial conditions for problem
user_timestep	function pointer to set the next time step
user_preprocess	function pointer to setup a preprocess operation
user_solve	function pointer to solve the system of equations
user_setparams	function pointer to set the parameters in the problem (always override this)
user_discretize	function pointer to select discretization scheme for the problem
user_bcs	function pointer to evaluate boundary conditions for the problem
user_res	function pointer to evaluate non-linear residuals for the problem
user_precon	function pointer to perform a linear preconditioning operation
user_-postprocess	function pointer to setup a postprocess operation
user_reset	function pointer to reset stateful data for next simulation
dat	pointer to the FINCH_DATA structure
param_data	user supplied pointer to a data structure needed in user_setparams

6.30.3.28 int uAverage (FINCH_DATA * *dat*)

Function integrates the conserved quantity to reture it's average in the domain.

6.30.3.29 int uTotal (FINCH_DATA * *dat*)

Function integrates the conserved quantity to return it's total in the domain.

6.30.3.30 int vanAlbada_discretization (const void * *user_data*)

Van Albada Discretization function for FINCH.

The van Albada discretization function for FINCH assumes the user_data parameter is the FINCH_DATA structure and discretizes the time and space portion of the problem with 2nd order finite differences and uses the van Albada slope limiter function to stabilize the advective physics.

6.31 flock.h File Reference

FundamentaL Off-gas Collection of Kernels.

```
#include "macaw.h"
#include "egret.h"
#include "finch.h"
#include "lark.h"
#include "skua.h"
#include "scopsowl.h"
#include "magpie.h"
```

6.31.1 Detailed Description

FundamentaL Off-gas Collection of Kernels. This is just a .h file that holds all the includes necessary to develop and run simulations for adsorption and/or mass/energy transfer problems for gaseous systems. Include this file into any other project or source code that needs the methods below.

Files Included in FLOCK

macaw.h egret.h finch.h lark.h skua.h scopsowl.h magpie.h

Author

Austin Ladshaw

Date

04/28/2014

Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file flock.h.

6.32 FlowProperties.h File Reference

Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics.

```
#include "Material.h"
#include "flock.h"
```

Classes

- class FlowProperties

  FlowProperties class object inherits from Material object.

Macros

- #define _gas_const 8.3144621

  Gas Law Constant - J/K/mol.

Functions

- template<>
  InputParameters validParams< FlowProperties > ()

6.32.1 Detailed Description

Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics. This file creates a material property object for the flow properties in the fixed-bed column. The flow properties are calculated based on some dimensional analysis, empirical relationships, and kinetic theory of gases. Those properties are then coupled the with mass and energy kernels in the domain to simulate the dynamic and non-linear behavior in the system.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file FlowProperties.h.

6.32.2 Macro Definition Documentation

6.32.2.1 #define _gas_const 8.3144621

Gas Law Constant - J/K/mol.

Definition at line 45 of file FlowProperties.h.

6.32.3 Function Documentation

6.32.3.1 template<> InputParameters validParams< FlowProperties > ( )

6.33 GAdvection.h File Reference

Kernel for use with the corresponding DGAdvection object.

```
#include "Kernel.h"
```

Classes

- class GAdvection

  GAdvection class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< GAdvection > ()

6.33.1 Detailed Description

Kernel for use with the corresponding DGAdvection object. This file creates a standard MOOSE kernel that is to be used in conjunction with the DGAdvection kernel for the discontinous Galerkin formulation of advection physics in MOOSE. In order to complete the DG formulation of the advective physics, this kernel must be utilized with every variable that also uses the DGAdvection kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file GAdvection.h.

6.33.2 Function Documentation

6.33.2.1 template<> InputParameters validParams< GAdvection > ( )

6.34 GAnisotropicDiffusion.h File Reference

Kernel for use with the corresponding DGAnisotropicDiffusion object.

```
#include "Kernel.h"
```

Classes

- class GAnisotropicDiffusion

  GAnisotropicDiffusion class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< GAnisotropicDiffusion > ()

6.34.1 Detailed Description

Kernel for use with the corresponding DGAnisotropicDiffusion object. This file creates a standard MOOSE kernel that is to be used in conjunction with the DGAnisotropicDiffusion kernel for the discontinous Galerkin formulation of advection physics in MOOSE. In order to complete the DG formulation of the advective physics, this kernel must be utilized with every variable that also uses the DGAAnisotropicDiffusion kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file GAnisotropicDiffusion.h.

6.34.2 Function Documentation

6.34.2.1 template<> InputParameters validParams< GAnisotropicDiffusion > ( )

6.35 GColumnHeatAdvection.h File Reference

Kernel for use with the corresponding DGColumnHeatAdvection object.

```
#include "GAdvection.h"
```

Classes

- class GColumnHeatAdvection

  GColumnHeatAdvection class object inherits from GAdvection object.

Functions

- template<>
  InputParameters validParams< GColumnHeatAdvection > ()

6.35.1 Detailed Description

Kernel for use with the corresponding DGColumnHeatAdvection object. This file creates a standard MOOSE kernel that is to be used in conjunction with DGColumnHeatAdvection for the discontinous Galerkin formulation of the heat advection physics for a fixed-bed adsorber. It couples with material properties to override the velocity parameter in the inherited GAdection kernel, then simply calls the corresponding methods of the base class.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file GColumnHeatAdvection.h.

6.35.2 Function Documentation

6.35.2.1 template<> InputParameters validParams< GColumnHeatAdvection >( )

6.36 GColumnHeatDispersion.h File Reference

Kernel for use with the corresponding DGColumnHeatDispersion object.

```
#include "GAnisotropicDiffusion.h"
```

Classes

- class GColumnHeatDispersion

  GColumnHeatDispersion class object inherits from GAnisotropicDiffusion object.

Functions

- template<>
  InputParameters validParams< GColumnHeatDispersion > ()

6.36.1 Detailed Description

Kernel for use with the corresponding DGColumnHeatDispersion object. This file creates a standard MOOSE kernel that is to be used in conjunction with the DGColumnHeatDispersion kernel for the discontinous Galerkin formulation of heat dispersion in a fixed-bed adsorber. This kernel is coupled with material properties, then uses that information to override the Diffusion parameter of the base class and call its methods.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file GColumnHeatDispersion.h.

6.36.2 Function Documentation

6.36.2.1 template<> InputParameters validParams< GColumnHeatDispersion > ( )

6.37 GColumnMassAdvection.h File Reference

Kernel for use with the corresponding DGColumnMassAdvection object.

```
#include "GAdvection.h"
```

Classes

- class GColumnMassAdvection

  GColumnMassAdvection class object inherits from GAdvection object.

Functions

- template<>
  InputParameters validParams< GColumnMassAdvection > ()

6.37.1 Detailed Description

Kernel for use with the corresponding DGColumnMassAdvection object. This file creates a standard MOOSE kernel that is to be used in conjunction with DGColumnMassAdvection for the discontinous Galerkin formulation of the mass advection physics for a fixed-bed adsorber. It couples with material properties to override the velocity parameter in the inherited GAdection kernel, then simply calls the corresponding methods of the base class.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file GColumnMassAdvection.h.

6.37.2 Function Documentation

6.37.2.1 template<> InputParameters validParams< GColumnMassAdvection > ( )

6.38 GColumnMassDispersion.h File Reference

Kernel for use with the corresponding DGColumnMassDispersion object.

```
#include "GAnisotropicDiffusion.h"
```

Classes

- class GColumnMassDispersion
  GColumnMassDispersion class object inherits from GAnisotropicDiffusion object.

Functions

- template<>
  InputParameters validParams< GColumnMassDispersion > ()

6.38.1 Detailed Description

Kernel for use with the corresponding DGColumnMassDispersion object. This file creates a standard MOOSE kernel that is to be used in conjunction with the DGColumnMassDispersion kernel for the discontinous Galerkin formulation of mass dispersion in a fixed-bed adsorber. This kernel is coupled with material properties, then uses that information to override the Diffusion parameter of the base class and call its methods.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file GColumnMassDispersion.h.

6.38.2 Function Documentation

6.38.2.1 template<> InputParameters validParams< GColumnMassDispersion > ( )

6.39 lark.h File Reference

Linear Algebra Residual Kernels.

```
#include "macaw.h"
#include <float.h>
```

Classes

- struct ARNOLDI_DATA
  Data structure for the construction of the Krylov subspaces for a linear system.
- struct GMRESLP_DATA
  Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.
- struct GMRESRP_DATA
  Data structure for the Restarted GMRES algorithm with Right Preconditioning.
- struct PCG_DATA
  Data structure for implementation of the PCG algorithms for symmetric linear systems.
- struct BiCGSTAB_DATA
  Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.
- struct CGS_DATA
  Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.
- struct OPTRANS_DATA
  Data structure for implementation of linear operator transposition.
- struct GCR_DATA
  Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.
- struct GMRESR_DATA
  Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)
- struct KMS_DATA
  Data structure for the implemenation of the Krylov Multi-Space (KMS) Method.
- struct PICARD_DATA
  Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.
- struct BACKTRACK_DATA
  Data structure for the implementation of Backtracking Linesearch.
- struct PJFNK_DATA
  Data structure for the implementation of the PJFNK algorithm for non-linear systems.
- struct NUM_JAC_DATA
  Data structure to form a numerical jacobian matrix with finite differences.

Macros

- #define MIN_TOL 1e-15
  Minimum Allowable Tolerance for linear and non-linear problems.

Enumerations

- enum krylov_method {
  GMRESLP, PCG, BiCGSTAB, CGS,
  FOM, GMRESRP, GCR, GMRESR }
  Enum of definitions for linear solver types in PJFNK.

Functions

- int update_arnoldi_solution (Matrix< double > &x, Matrix< double > &x0, ARNOLDI_DATA *arnoldi_dat)

  Function to update the linear vector x based on the Arnoldi Krylov subspace.
- int arnoldi (int(*matvec)(const Matrix< double > &v, Matrix< double > &w, const void *data), int(*precon)(const Matrix< double > &b, Matrix< double > &p, const void *data), Matrix< double > &r0, ARNOLDI_DATA *arnoldi_dat, const void *matvec_data, const void *precon_data)

  Function to factor a linear operator into an orthonormal basis and upper Hessenberg matrix.
- int gmresLeftPreconditioned (int(*matvec)(const Matrix< double > &v, Matrix< double > &w, const void *data), int(*precon)(const Matrix< double > &b, Matrix< double > &p, const void *data), Matrix< double > &b, GMRESLP_DATA *gmreslp_dat, const void *matvec_data, const void *precon_data)

  Function to iteratively solve a non-symmetric, indefinite linear system with GMRESLP.
- int fom (int(*matvec)(const Matrix< double > &v, Matrix< double > &w, const void *data), int(*precon)(const Matrix< double > &b, Matrix< double > &p, const void *data), Matrix< double > &b, GMRESLP_DATA *gmreslp_dat, const void *matvec_data, const void *precon_data)

  Function to directly solve a non-symmetric, indefinite linear system with FOM.
- int gmresRightPreconditioned (int(*matvec)(const Matrix< double > &v, Matrix< double > &w, const void *data), int(*precon)(const Matrix< double > &b, Matrix< double > &p, const void *data), Matrix< double > &b, GMRESRP_DATA *gmresrp_dat, const void *matvec_data, const void *precon_data)

  Function to iteratively solve a non-symmetric, indefinite linear system with GMRESRP.
- int pcg (int(*matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void *data), int(*precon)(const Matrix< double > &r, Matrix< double > &z, const void *data), Matrix< double > &b, PCG_DATA *pcg_dat, const void *matvec_data, const void *precon_data)

  Function to iteratively solve a symmetric, definite linear system with PCG.
- int bicgstab (int(*matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void *data), int(*precon)(const Matrix< double > &r, Matrix< double > &z, const void *data), Matrix< double > &b, BiCGSTAB_DATA *bicg_dat, const void *matvec_data, const void *precon_data)

  Function to iteratively solve a non-symmetric, definite linear system with BiCGSTAB.
- int cgs (int(*matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void *data), int(*precon)(const Matrix< double > &r, Matrix< double > &z, const void *data), Matrix< double > &b, CGS_DATA *cgs_dat, const void *matvec_data, const void *precon_data)

  Function to iteratively solve a non-symmetric, definite linear system with CGS.
- int operatorTranspose (int(*matvec)(const Matrix< double > &v, Matrix< double > &Av, const void *data), Matrix< double > &r, Matrix< double > &u, OPTRANS_DATA *transpose_dat, const void *matvec_data)

  *Function that is used to perform transposition of a linear operator and results in a new vector A^T*r=u.*
- int gcr (int(*matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *data), int(*precon)(const Matrix< double > &r, Matrix< double > &Mr, const void *data), Matrix< double > &b, GCR_DATA *gcr_dat, const void *matvec_data, const void *precon_data)

  Function to iteratively solve a non-symmetric, definite linear system with GCR.
- int gmresrPreconditioner (const Matrix< double > &r, Matrix< double > &Mr, const void *data)

  Function used in conjunction with GMRESR to apply GMRESRP iterations as a preconditioner.
- int gmresr (int(*matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *data), int(*terminal_precon)(const Matrix< double > &r, Matrix< double > &Mr, const void *data), Matrix< double > &b, GMRESR_DATA *gmresr_dat, const void *matvec_data, const void *term_precon_data)

  Function to iteratively solve a non-symmetric, indefinite linear system with GMRESR.
- int kmsPreconditioner (const Matrix< double > &r, Matrix< double > &Mr, const void *data)

  Preconditioner function for the Krylov Multi-Space.
- int krylovMultiSpace (int(*matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void *data), int(*terminal_precon)(const Matrix< double > &r, Matrix< double > &Mr, const void *data), Matrix< double > &b, KMS_DATA *kms_dat, const void *matvec_data, const void *term_precon_data)

  Function to iteratively solve a non-symmetric, indefinite linear system with KMS.
- int picard (int(*res)(const Matrix< double > &x, Matrix< double > &r, const void *data), int(*evalx)(const Matrix< double > &x0, Matrix< double > &x, const void *data), Matrix< double > &x, PICARD_DATA *picard_dat, const void *res_data, const void *evalx_data)

Function to iteratively solve a non-linear system using the Picard or Fixed-Point method.

- int jacvec (const Matrix< double > &v, Matrix< double > &Jv, const void *data)

  Function to form a linear operator of a Jacobian matrix used along with the PJFNK method.

- int backtrackLineSearch (int(*feval)(const Matrix< double > &x, Matrix< double > &F, const void *data), Matrix< double > &Fkp1, Matrix< double > &xkp1, Matrix< double > &pk, double normFk, BACKTRACK_DATA *backtrack_dat, const void *feval_data)

  Function to perform a Backtracking Line Search operation to smooth out convergence of PJFNK.

- int pjfnk (int(*res)(const Matrix< double > &x, Matrix< double > &F, const void *data), int(*precon)(const Matrix< double > &r, Matrix< double > &p, const void *data), Matrix< double > &x, PJFNK_DATA *pjfnk_dat, const void *res_data, const void *precon_data)

  Function to perform the PJFNK algorithm to solve a non-linear system of equations.

- int NumericalJacobian (int(*Func)(const Matrix< double > &x, Matrix< double > &F, const void *user_data), const Matrix< double > &x, Matrix< double > &J, int Nx, int Nf, NUM_JAC_DATA *jac_dat, const void *user_data)

  Function to form a full numerical Jacobian matrix from a given non-linear function.

6.39.1 Detailed Description

Linear Algebra Residual Kernels. lark.cpp

The functions contained within are designed to solve generic linear and non-linear square systems of equations given a function argument and data from the user. Optionally, the user can also provide a function to return a preconditioning result that will be applied to the system.

Having the user define how the preconditioning is carried out provides two major advantages: (1) we do not need to store and large, sparse preconditioning matrices and instead only store the preconditioned vector result and (2) this allows the user to use any kind of preconditioner they see fit for their problem.

The Arnoldi function is typically not called by the user, but can be if desired. It accepts the function arguments and a residual vector to form an orthonormal basis of the Krylov subspace using the Modified Gram-Schmidt process (aka Arnoldi Iteration). This function is called by GMRES to iteratively solve a linear system of equations. Note that you can use this function to directly solve the linear system as long as that system is not too large. Construction of the basis is expensive, which is why this is used as a sub-function of an iterative method.

The Restarted GMRES function will accept function arguments for a linear system and attempt to solve said system iteratively by constructing an orthonormal basis from the Krylov function. Note that this GMRES function does support restarting and will use restarting by default if the linear system is too large.

Also included is a GMRES algorithm without restarting. This will directly solve the linear system within residual tolerance using a Full Orthogonal basis set of that system. It is equivalent to calling the Krylov method with the k parameter equal to N (i.e. the number of equations). This method is nick-named the Full Othogonalization Method (FOM), although the true FOM algorithm in literature is slightly different.

The PJFNK function will accept function arguments for a square, non-linear system of equations and attempt to solve it iteratively using both the GMRES and Krylov functions with Newton's method to convert the non-linear system into a linear system.

Also built here is a PCG implementation for solving symmetric linear systems. Can also be called by PJFNK if we know that the linear system (i.e. the Jacobian) is symmetric. This algorithm is significantly more efficient than GMRES, but is only valid if the system of equations is symmetric.

Other linear solvers implemented in this work are the BiCGSTAB and CGS algorithms for non-symmetric, positive definite matrices. These algorithms are significantly more computationally efficient than GMRES or FOM. However, they can both break down if the linear system is poorly conditioned. In general, you only want to use these methods if you have preconditioning available and your linear system is very, very large. Otherwise, you will be better suited to using GMRES or FOM.

There is also an implementation of the Generalized Conjugate Residual (GCR) method with and without restarting. This is a GMRES-like method that should give the exact solution within N iterations, where N is the original size of the matrix. Built ontop of the GCR method is a GMRESR (or GMRES Recursive) algorithm that uses GCR as the base method and performs GMRESRP iterations as a preconditioner at each iteration of GCR. This is the only linear

solver that has built-in preconditioning. As a result, it may be slower than other algorithms for simple problems, but generally will have much better convergence behavior and will almost always give better residual reduction, even for hard to solve problems.

We have also developed a novel/experimental iterative method based on the idea of recursively preconditioning a Krylov Subspace with more Krylov Subspaces. We have called with algorithm the Krylov Multi-Space (KMS) method. This algorithm is based on publications from Vorst and Vuik (1991) and Saad (1993). The idea is too use the FGMRES algorithm developed by Saad (1993) and precondition it with more FGMRES steps, i.e., nesting the iterations as Vorst and Vuik (1991) had proposed. In this way, we have created a generalized Krylov Subspace method that has it's own variable preconditioner that can be adjusted depending on the user's desired complexity and convergence rate. If the levels of recursion requested is zero, then this algorithm is exactly equal to GMRES with right preconditioning. If the level is one, then it is FGMRES with a GMRES preconditioner. However, we allow the levels of recursion to reach up to 5, thus allowing us to precondition the preconitioners with more GMRES steps. This can result is significantly faster convergence rates, but is typically only necessary for very large or difficult to solve problems.

NOTE: There are three GMRES implementations: (i) gmresLP, (ii) fom, and (iii) gmresRP. GMRESLP is a restarted GMRES implementation that is left preconditioned and only checks the residual on the outer loops. This may be less efficient than GMRESRP, which can check both outer and inner loop residuals. However, GMRESRP has to use right preconditioning, which also slightly changes the convergence behavior of the linear system. GMRES with left preconditioning and without restarting will just build the full subspace by default, thus solving the system exactly, but may require too much memory. You can do a GMRESRP unrestarted by specifying that the restart parameter be equal to the size of the problem.

Basic Implementation Details:

Linear Solvers -> Solve Ax=b for x

Non-Linear Solvers -> Solve F(x)=0 for x

All implementations require system size to be 2 or greater

Author

Austin Ladshaw

Date

10/14/2014

Copyright

Definition in file lark.h.

6.39.2 Macro Definition Documentation

6.39.2.1 #define MIN_TOL 1e-15

Minimum Allowable Tolerance for linear and non-linear problems.

Definition at line 111 of file lark.h.

6.39.3 Enumeration Type Documentation

6.39.3.1 enum krylov_method

Enum of definitions for linear solver types in PJFNK.

Enum delineates the available Krylov Subspace methods that can be used to solve the linear sub-problem at each non-linear iteration in a Newton method.

Enumerator

GMRESLP

PCG

BiCGSTAB

CGS

FOM

GMRESRP

GCR

GMRESR

Definition at line 492 of file lark.h.

6.39.4 Function Documentation

6.39.4.1 int arnoldi (int(∗)(const Matrix< double > &v, Matrix< double > &w, const void ∗data) *matvec,* int(∗)(const Matrix< double > &b, Matrix< double > &p, const void ∗data) *precon,* Matrix< double > & *r0,* ARNOLDI_DATA ∗ *arnoldi_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to factor a linear operator into an orthonormal basis and upper Hessenberg matrix.

This function performs the Arnoldi algorithm to factor a linear operator into an orthonormal basis and upper Hessenberg matrix. Each orthonormal vector is formed using a Modified Gram-Schmidt procedure. When used in conjunction with GMRESLP, user may supply a preconditioning operator to improve convergence of the linear system. However, this function can be used by itself to factor the user's linear operator.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
r0	user supplied vector to serve as the first basis vector in the orthonormal basis
arnoldi_dat	pointer to the ARNOLDI_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.2 int backtrackLineSearch (int(∗)(const Matrix< double > &x, Matrix< double > &F, const void ∗data) *feval,* Matrix< double > & *Fkp1,* Matrix< double > & *xkp1,* Matrix< double > & *pk,* double *normFk,* BACKTRACK_DATA ∗ *backtrack_dat,* const void ∗ *feval_data*)

Function to perform a Backtracking Line Search operation to smooth out convergence of PJFNK.

This function performs a simple backtracking line search operation on the residuals from the PJFNK method. The step size of the non-linear iteration is checked against a level of tolerance for residual reduction, then adjusted down if necessary. This method always starts out with the maximum allowable step size. If the largest step size is fine, then the algorithm does nothing. Otherwise, it iteratively adjusts the step size down, until a suitable step is found. In the case that no suitable step is found, this algorithm will report failure to the PJFNK method and PJFNK will decide whether to continue trying to find a global minimum or report that it is stuck in a local minimum.

Parameters

feval	user supplied residual function for the non-linear system
Fkp1	vector holding the residuals for the next non-linear step
xkp1	vector holding the solution for the next non-linear step
pk	vector holding the current non-linear search direction
normFk	value of the current non-linear residual
backtrack_dat	pointer to the BACKTRACK_DATA data structure
feval_data	user supplied void pointer to the data structure needed for residual evaluation

Note

int (∗feval) (const Matrix<double>& x, Matrix<double> &F, const void ∗data)

This is a user supplied function for the non-linear residuals. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix x representing the current non-linear variables. Those variables are used to evaluate the users functions and return the residuals in the matrix F. The void pointer data is a data structure provided by the user to hold information the function may need in order to form the residuals.

6.39.4.3 int bicgstab (int(∗)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &z, const void ∗data) *precon,* Matrix< double > & *b,* BiCGSTAB_DATA ∗ *bicg_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to iteratively solve a non-symmetric, definite linear system with BiCGSTAB.

This function iteratively solves a non-symmetric, definite linear system using the Bi-Conjugate Gradient STABilized (BiCGSTAB) method. This is a highly efficient algorithm for solving non-symmetric problems, but will occassionally breakdown and fail. Most common failures are caused by poor preconditioning. Works very well for grid-based linear systems.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
bicg_dat	pointer to the BiCGSTAB_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified

the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.4 int cgs (int(∗)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &z, const void ∗data) *precon,* Matrix< double > & *b,* CGS_DATA ∗ *cgs_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to iteratively solve a non-symmetric, definite linear system with CGS.

This function iteratively solves a non-symmetric, definite linear system using the Conjugate Gradient Squared (CGS) method. This is an extremely efficient algorithm for solving non-symmetric problems, but will often breakdown and fail. Most common failures are caused by poor or no preconditioning. Works very will for grid-based linear systems.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
cgs_dat	pointer to the CGS_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.5 int fom (int(∗)(const Matrix< double > &v, Matrix< double > &w, const void ∗data) *matvec,* int(∗)(const Matrix< double > &b, Matrix< double > &p, const void ∗data) *precon,* Matrix< double > & *b,* GMRESLP_DATA ∗ *gmreslp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to directly solve a non-symmetric, indefinite linear system with FOM.

This function directly solves a non-symmetric, indefinite linear system using the Full Orthogonalization Method (FOM). This algorithm is exactly equivalent to GMRESLP without restarting. Therefore, it uses the GMRESLP_DATA structure and calls the GMRESLP algorithm without using restarts. As a result, it never checks linear residuals. However, this should give the exact solution upon completion, assuming the linear operator is not singular.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmreslp_dat	pointer to the GMRESLP_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.6 int gcr (int(∗)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &Mr, const void ∗data) *precon,* Matrix< double > & *b,* GCR_DATA ∗ *gcr_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to iteratively solve a non-symmetric, definite linear system with GCR.

This function iteratively solves a non-symmetric, definite linear system using the Generalized Conjugate Residual (GCR) method. Similar to GMRESRP, this algorithm will construct a growing orthonormal basis set that will eventually form the exact solution to the linear system. However, this algorithm is less efficient than GMRESRP and can suffer breakdowns if the linear system is indefinite.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gcr_dat	pointer to the GCR_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear

operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.7 int gmresLeftPreconditioned (int(∗)(const Matrix< double > &v, Matrix< double > &w, const void ∗data) *matvec,* int(∗)(const Matrix< double > &b, Matrix< double > &p, const void ∗data) *precon,* Matrix< double > & *b,* GMRESLP_DATA ∗ *gmreslp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESLP.

This function iteratively solves a non-symmetric, indefinite linear system using the Generalized Minimum RESidual method with Left Preconditioning (GMRESLP). It calls the Arnoldi algorithm to factor a linear operator into an orthonormal basis and upper Hessenberg matrix, then uses that factorization to form an approximation to the linear system. Because this algorithm uses left-side preconditioning, it can only check the linear residuals at the outer iterations.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmreslp_dat	pointer to the GMRESLP_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.8 int gmresr (int(∗)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &Mr, const void ∗data) *terminal_precon,* Matrix< double > & *b,* GMRESR_DATA ∗ *gmresr_dat,* const void ∗ *matvec_data,* const void ∗ *term_precon_data*)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESR.

This function iteratively solves a non-symmetric, indefinite linear system using the Generalized Minimum RESidual Recursive (GMRESR) method. This algorithm actually uses GCR at the outer iterations, but stabilizes GCR with GMRESRP inner iterations to implicitly form a variable preconditioner to the linear system. As such, this is one of only two methods that inherently includes preconditioning (the other is KMS), without any user supplied preconditioning operator. However, this algorithms is signficantly more computationally expensive than GCR or GMRESRP separately. It should only be used for solving very large or very hard to solve linear systems.

Parameters

matvec	user supplied linear operator given as an int function
terminal_precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmresr_dat	pointer to the GMRESR_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
term_precon_-data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗terminal_precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.9 int gmresRightPreconditioned (int(∗)(const Matrix< double > &v, Matrix< double > &w, const void ∗data) *matvec,* int(∗)(const Matrix< double > &b, Matrix< double > &p, const void ∗data) *precon,* Matrix< double > & *b,* GMRESRP_DATA ∗ *gmresrp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESRP.

This function iteratively solves a non-symmetric, indefinite linear system using the Generalized Minimum RESidual method with Right Preconditioning (GMRESRP). Because this algorithm uses right preconditioning, it is able to check the linear residuals at both the outer and inner iterations. This may be much for efficient compared to G-MRESLP. In order to check inner residuals, this algorithm has to perform it's own internal Modified Gram-Schmidt procedure and will not call the Arnoldi algorithm.

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmresrp_dat	pointer to the GMRESRP_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.10 int gmresrPreconditioner (const Matrix< double > & *r,* Matrix< double > & *Mr,* const void * *data*)

Function used in conjunction with GMRESR to apply GMRESRP iterations as a preconditioner.

This function is required to take the form of the user supplied preconditioning functions for other iterative methods. However, it cannot be used in conjunction with any other Krylov method. It is only called by the GMRESR function when the preconditioner needs to be applied.

Parameters

r	vector supplied to the preconditioner to operate on
Mr	vector to hold the result of the preconditioning operation
data	void pointer to the GMRESR_DATA data structure

6.39.4.11 int jacvec (const Matrix< double > & *v,* Matrix< double > & *Jv,* const void * *data*)

Function to form a linear operator of a Jacobian matrix used along with the PJFNK method.

This function is used in conjunction with the PJFNK routine to form a linear operator that a Krylov method can operate on. This linear operator is formed from the current residual vector of the non-linear iteration in PJFNK using a finite difference approximation.

Jacobian Linear Operator: J*v = (F(x_k + eps*v) - F(x_k)) / eps

Parameters

v	vector to be multiplied by the Jacobian matrix
Jv	storage vector for the result of the Jacobi-vector product
data	void pointer to the PJFNK_DATA data structure holding solver information

6.39.4.12 int kmsPreconditioner (const Matrix< double > & *r,* Matrix< double > & *Mr,* const void * *data*)

Preconditioner function for the Krylov Multi-Space.

This function is required to take the form of the user supplied preconditioning functions for other iterative methods. However, it cannot be used in conjunction with any other Krylov method. It is only called by the KMS function when the preconditioner needs to be applied.

Parameters

r	vector supplied to the preconditioner to operate on
Mr	vector to hold the result of the preconditioning operation
data	void pointer to the KMS_DATA data structure

6.39.4.13 int krylovMultiSpace (int(*)(const Matrix< double > &x, Matrix< double > &Ax, const void *data) *matvec,* int(*)(const Matrix< double > &r, Matrix< double > &Mr, const void *data) *terminal_precon,* Matrix< double > & *b,* KMS_DATA * *kms_dat,* const void * *matvec_data,* const void * *term_precon_data*)

Function to iteratively solve a non-symmetric, indefinite linear system with KMS.

This function iteratively solves a non-symmetric, indefinite linear system using the Krylov Multi-Space (KMS) method. This algorithm uses GMRESRP at both outer and inner iterations to implicitly form a variable preconditioner

to the linear system. As such, this is one of only two methods that inherently includes preconditioning, without any user supplied preconditioning operator (the other being GMRESR). The advantage to this method over GMRESR is that this method is GMRES at its core, and will therefore never breakdown or need to be stabilized. Additionally, you can call this method and set it's max_level parameter (see KMS_DATA) to 0, which will make this algorithm exactly equal to GMRESRP. If the max_level is set to 1, then this algorithm is exactly FGMRES (Saad, 1993) with the GMRES algorithm as a preconditioner. However, you can set max_level higher to precondition the preconditioners with more preconditioners. Thus creating a method with any desired complexity or rate of convergence.

Parameters

matvec	user supplied linear operator given as an int function
terminal_precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
kms_dat	pointer to the KMS_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
term_precon_-data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗terminal_precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.14 int NumericalJacobian (int(∗)(const Matrix< double > &x, Matrix< double > &F, const void ∗user_data) *Func,* const Matrix< double > & *x,* Matrix< double > & *J,* int *Nx,* int *Nf,* NUM_JAC_DATA ∗ *jac_dat,* const void ∗ *user_data*)

Function to form a full numerical Jacobian matrix from a given non-linear function.

This function uses finite differences to form a full rank Jacobian matrix for a user supplied non-linear function. The Jacobian matrix will be formed at the current state of the non-linear variables x and stored in a full matrix J. Integers Nx and Nf are used to determine the size of the Jacobian matrix.

Parameters

Func	user supplied function for evaluation of the non-linear system
x	matrix holding the current value of the non-linear variables
J	matrix that will store the numerical Jacobian result
Nx	number of non-linear variables in the system
Nf	number of non-linear functions in the system
jac_dat	pointer to the NUM_JAC_DATA data structure
user_data	user supplied void pointer to a data structure used in the non-linear function

6.39.4.15 int operatorTranspose (int(∗)(const Matrix< double > &v, Matrix< double > &Av, const void ∗data) *matvec,* Matrix< double > & *r,* Matrix< double > & *u,* OPTRANS_DATA ∗ *transpose_dat,* const void ∗ *matvec_data*)

Function that is used to perform transposition of a linear operator and results in a new vector A^T∗r=u.

This function takes a user supplied linear operator and forms the result of that operator transposed and multiplied by a given vector r (A^T∗r=u). Transposition is accomplised by reordering the transpose operator and multiplying the non-transposed operator by a complete set of orthonormal vectors. The end result gives the ith component of the vector u for each operation (u_i = r^T∗A∗i). Here, i is a vector made from the ith column of the identity matrix. If the linear system if sufficiently large, then this operation may take some time.

Parameters

matvec	user supplied linear operator given as an int function
r	vector to be multiplied by the transpose of the operator
u	vector to store the result of the operator transposition (u=A^T∗r)
transpose_dat	pointer to the OPTRANS_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

6.39.4.16 int pcg (int(∗)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &z, const void ∗data) *precon,* Matrix< double > & *b,* PCG_DATA ∗ *pcg_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

Function to iteratively solve a symmetric, definite linear system with PCG.

This function iteratively solves a symmetric, definite linear system using the Preconditioned Conjugate Gradient (PCG) method. The PCG algorithm is optimal in terms of efficiency and residual reduction, but only if the linear system is symmetric. PCG will fail if the linear operator is non-symmetric!

Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
pcg_dat	pointer to the PCG_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

Note

int (∗matvec) (const Matrix<double>& v, Matrix<double> &Av, const void ∗data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.17 **int picard (int(∗)(const Matrix< double > &x, Matrix< double > &r, const void ∗data) *res,* int(∗)(const Matrix< double > &x0, Matrix< double > &x, const void ∗data) *evalx,* Matrix< double > & *x,* PICARD_DATA ∗ *picard_dat,* const void ∗ *res_data,* const void ∗ *evalx_data*)**

Function to iteratively solve a non-linear system using the Picard or Fixed-Point method.

This function iteratively solves a non-linear system using the Picard method. User supplies a residual function and a weak solution form function. The weak form function is used to approximate the next solution vector for the non-linear system and the residual function is used to determine convergence. User also supplies an initial guess to the non-linear system as a matix x, which will also be used to store the solution. This algorithm is very simple and may not be sufficient to solve complex non-linear systems.

Parameters

res	user supplied function for the non-linear residuals of the system
evalx	user supplied function for the weak form to estimate the next solution
x	user supplied matrix holding the initial guess to the non-linear system
picard_dat	pointer to the PICARD_DATA data structure
res_data	user supplied void pointer to a data structure used for residual evaluations
evalx_data	user supplied void pointer to a data structure used for evaluation of weak form

Note

int (∗res) (const Matrix<double>& x, Matrix<double> &F, const void ∗data)

This is a user supplied function for the non-linear residuals. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix x representing the current non-linear variables. Those variables are used to evaluate the users functions and return the residuals in the matrix F. The void pointer data is a data structure provided by the user to hold information the function may need in order to form the residuals.

---

int (∗evalx) (const Matrix<double>& x0, Matrix<double> &x, const void ∗data)

This is a user supplied function to approximate the next solution vector x based on the previous solution vector x0. The x0 matrix is passed to this function and must be used to edit the entries of x based on the weak form of the problem. The user is free to define any weak form approximation. Void pointer data is the users data structure that may be used to pass additional information into this function in order to evaluate the weak form.

Example Residual: F(x) = x^2 + x - 1 Goal is to make this function equal zero

Example Weak Form: x = 1 - x0^2 Rearrage residual to form a weak solution

---

6.39.4.18 **int pjfnk (int(∗)(const Matrix< double > &x, Matrix< double > &F, const void ∗data) *res,* int(∗)(const Matrix< double > &r, Matrix< double > &p, const void ∗data) *precon,* Matrix< double > & *x,* PJFNK_DATA ∗ *pjfnk_dat,* const void ∗ *res_data,* const void ∗ *precon_data*)**

Function to perform the PJFNK algorithm to solve a non-linear system of equations.

This function solves a non-linear system of equations using the Preconditioned Jacobian- Free Newton-Krylov (P-JFNK) algorithm. Each non-linear step of this method results in a linear sub-problem that is solved iteratively with one of the Krylov methods in the krylov_method enum. User must supplied a residual function that computes the

non-linear residuals of the system given the current state of the variables x. Additionally, the user must also supplied an initial guess to the non-linear system. Optionally, the user may supply a preconditioning function for the linear sub-problem.

Basic Steps: (i) Calc F(x_k), (ii) Solve J(x_k)∗s_k=-F(x_k) for s_k, (iii) Form x_kp1 = x_k + s_k

Parameters

res	user supplied residual function for the non-linear system
precon	user supplied preconditioning function for the linear sub-problems
x	user supplied initial guess and storage location of the solution
pjfnk_dat	pointer to the PJFNK_DATA data structure
res_data	user supplied void pointer to data structure used in residual function
precon_data	user supplied void pointer to data structure used in preconditioning function

Note

int (∗res) (const Matrix<double>& x, Matrix<double> &F, const void ∗data)

This is a user supplied function for the non-linear residuals. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix x representing the current non-linear variables. Those variables are used to evaluate the users functions and return the residuals in the matrix F. The void pointer data is a data structure provided by the user to hold information the function may need in order to form the residuals.

---

int (∗precon) (const Matrix<double>& b, Matrix<double> &Mb, const void ∗data)

This is a user supplied function for a preconditioning operator. It has the same form as the linear operators from the Krylov methods and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the jacvec linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

6.39.4.19 int update_arnoldi_solution (Matrix< double > & *x*, Matrix< double > & *x0*, ARNOLDI_DATA ∗ *arnoldi_dat*)

Function to update the linear vector x based on the Arnoldi Krylov subspace.

This function will update a solution vector x based on the previous solution x0 given the orthonormal basis and upper Hessenberg matrix formed in the Arnoldi algorithm. Updating is automatically called by the GMRESLP function. It is expected that the Arnoldi algorithm has already been called prior to calling this function.

Parameters

x	matrix that will hold the new updated solution to the linear system
x0	matrix that holds the previous solution to the linear system
arnoldi_dat	pointer to the ARNOLDI_DATA data structure

6.40 LinearDrivingForce.h File Reference

Standard kernel for a generic coupled linear driving force mechanism.

```
#include "Kernel.h"
```

Classes

- class LinearDrivingForce

LinearDrivingForce class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< LinearDrivingForce > ()

6.40.1 Detailed Description

Standard kernel for a generic coupled linear driving force mechanism. This file creates a standard MOOSE kernel for a linear driving force type of mechanism that can be added to the non-linear residuals. It contains a boolean argument to determine whether the driving force is gaining or losing, a coefficient for the rate of the driving force, and a driving value to where the non-linear coupled variable is heading towards.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file LinearDrivingForce.h.

6.40.2 Function Documentation

6.40.2.1 template<> InputParameters validParams< LinearDrivingForce > ( )

6.41 macaw.h File Reference

MAtrix CAlculation Workspace.

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include <exception>
#include "error.h"
```

Classes

- class Matrix< T >

  Templated C++ Matrix Class Object (click Matrix to go to function definitions)

Macros

- #define M_PI 3.14159265358979323846264338327950288

  Value of PI with double precision.

6.41.1 Detailed Description

MAtrix CAlculation Workspace. macaw.cpp

This is a small C++ library that faciltates the use and construction of real matrices using vector objects. The Matrix class is templated so that users are able to work with matrices of any type including, but not limited to: (i) doubles, (ii) ints, (iii) floats, and (iv) even other matrices! Routines and functions are defined for Dense matrix operations. As a result, we typically only use Column Matrices (or Vectors) when doing any actual simulations. However, the development of this class was integral to the development and testing of the Sparse matrix operators in lark.h.

While the primary goal of this object was to define how to operate on real matrices, we could extend this idea to complex matrices as well. For this, we could develop objects that represent imaginary and complex numbers and then create a Matrix of those objects. For this reason, the matrix operations here are all templated to abstract away the specificity of the type of matrix being operated on.

Author

Austin Ladshaw

Date

01/07/2014

Copyright

Definition in file macaw.h.

6.41.2 Macro Definition Documentation

6.41.2.1 #define M_PI 3.14159265358979323846264338327950288

Value of PI with double precision.

Definition at line 43 of file macaw.h.

Referenced by Matrix< T >::IntegralTotal().

6.42 magpie.h File Reference

Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria.

```
#include "lmcurve.h"
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include "error.h"
```

Classes

- struct GSTA_DATA

  GSTA Data Structure.
- struct mSPD_DATA

  MSPD Data Structure.
- struct GPAST_DATA

  GPAST Data Structure.
- struct SYSTEM_DATA

  System Data Structure.
- struct MAGPIE_DATA

  MAGPIE Data Structure.

Macros

- #define DBL_EPSILON 2.2204460492503131e-016

  Machine precision value used for approximating gradients.
- #define Z 10.0

  Surface coordination number used in the MSPD activity model.
- #define A 3.13E+09

  Corresponding van der Waals standard area for our coordination number (cm^2/mol)
- #define V 18.92

  Corresponding van der Waals standard volume for our coordination number (cm^3/mol)
- #define Po 100.0

  Standard State Pressure - Units: kPa.
- #define R 8.3144621

  Gas Constant - Units: J/(K∗mol) = kB ∗ Na.
- #define Na 6.0221413E+23

  Avagadro's Number - Units: molecules/mol.
- #define kB 1.3806488E-23

  Boltzmann's Constant - Units: J/K.
- #define shapeFactor(v_i) (((Z - 2) ∗ v_i) / (Z ∗ V)) + (2 / Z)

  This macro replaces all instances of shapeFactor(#) with the following single line calculation.
- #define lnKo(H, S, T) -(H / (R ∗ T)) + (S / R)

  This macro calculates the natural log of the dimensionless isotherm parameter.
- #define He(qm, K1, m) (qm ∗ K1) / (m ∗ Po)

  This macro calculates the Henry's Coefficient for the ith component.

Functions

- double qo (double po, const void *data, int i)

  Function computes the result of the GSTA isotherm for the ith species.
- double dq_dp (double p, const void *data, int i)

  Function computes the derivative of the GSTA model with respect to partial pressure.
- double q_p (double p, const void *data, int i)

  Function computes the ratio between the adsorbed amount and partial pressure for the GSTA isotherm.
- double PI (double po, const void *data, int i)

  Function computes the spreading pressure integral of the ith species.
- double Qst (double po, const void *data, int i)

  Function computes the heat of adsorption based on the ith species GSTA parameters.
- double eMax (const void *data, int i)

  Function to approximate the maximum lateral energy term for the ith species.
- double lnact_mSPD (const double *par, const void *data, int i, volatile double PI)

  Function to evaluate the MSPD activity coefficient for the ith species.
- double grad_mSPD (const double *par, const void *data, int i)

  Function to approximate the derivative of the MSPD activity model with spreading pressure.
- double qT (const double *par, const void *data)

  Function to calculate the total adsorbed amount (mol/kg) for the mixed surface phase.
- void initialGuess_mSPD (double *par, const void *data)

  Function to provide an initial guess to the unknown parameters being solved for in GPAST.
- void eval_po_PI (const double *par, int m_dat, const void *data, double *fvec, int *info)

  Function used with lmfit to evaluate the reference state pressure of a species based on spreading pressure.
- void eval_po_qo (const double *par, int m_dat, const void *data, double *fvec, int *info)

  Function used with lmfit to evaluate the reference state pressure of a species based on that species isotherm.
- void eval_po (const double *par, int m_dat, const void *data, double *fvec, int *info)

  Function used with lmfit to evaluate the reference state pressure of a species based on a sub-system.
- void eval_eta (const double *par, int m_dat, const void *data, double *fvec, int *info)

  Function used with lmfit to evaluate the binary interaction parameters for each unique species pair.
- void eval_GPAST (const double *par, int m_dat, const void *data, double *fvec, int *info)

  Function used with lmfit to solve the GPAST system of equations.
- int MAGPIE (const void *data)

  Function to call all sub-routines to solve a MAGPIE/GPAST problem at a given temperature and pressure.

6.42.1 Detailed Description

Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria. magpie.cpp

This file contains all functions and routines associated with predicting isothermal adsorption equilibria from only single component isotherm information. The basis of the model is the Adsorbed Solution Theory developed by Myers and Prausnitz (1965). Added to that base model is a procedure by which we can predict the non-idealities present at the surface phase by solving a closed system of equations involving the activity model.

For more details on this procedure, check out our publication in AIChE where we give a fully feature explaination of our Generalized Predictive Adsorbed Solution Theory (GPAST).

Reference: Ladshaw, A., Yiacoumi, S., and Tsouris, C., "A generalized procedure for the prediction of multicomponent adsorption equilibria", AIChE J., vol. 61, No. 8, p. 2600-2610, 2015.

MAGPIE represents a special case of the more general GPAST procedure, wherin the isotherm for each species is respresent by the GSTA isotherm (see gsta_opt.h) and the activity model for non-ideality at the adsorbent surface is a Modified Spreading Pressure Dependent (MSPD) model. See the above paper reference for more details.

Author

Austin Ladshaw

Date

12/17/2013

Copyright

Definition in file magpie.h.

6.42.2 Macro Definition Documentation

6.42.2.1 #define A 3.13E+09

Corresponding van der Waals standard area for our coordination number (cm^2/mol)

Definition at line 56 of file magpie.h.

Referenced by Matrix< T >::inverse(), Matrix< T >::ladshawSolve(), and Matrix< T >::tridiagonalSolve().

6.42.2.2 #define DBL_EPSILON 2.2204460492503131e-016

Machine precision value used for approximating gradients.

Definition at line 48 of file magpie.h.

6.42.2.3 #define He(*qm, K1, m*) (qm * K1) / (m * Po)

This macro calculates the Henry's Coefficient for the ith component.

Definition at line 91 of file magpie.h.

6.42.2.4 #define kB 1.3806488E-23

Boltzmann's Constant - Units: J/K.

Definition at line 76 of file magpie.h.

6.42.2.5 #define lnKo(*H, S, T*) -(H / (R * T)) + (S / R)

This macro calculates the natural log of the dimensionless isotherm parameter.

Definition at line 86 of file magpie.h.

6.42.2.6 #define Na 6.0221413E+23

Avagadro's Number - Units: molecules/mol.

Definition at line 72 of file magpie.h.

6.42.2.7 #define Po 100.0

Standard State Pressure - Units: kPa.

Definition at line 64 of file magpie.h.

6.42.2.8 #define R 8.3144621

Gas Constant - Units: J/(K*mol) = kB * Na.

Definition at line 68 of file magpie.h.

6.42.2.9 #define shapeFactor(v_i) (((Z - 2) * v_i) / (Z * V)) + (2 / Z)

This macro replaces all instances of shapeFactor(#) with the following single line calculation.

Definition at line 81 of file magpie.h.

6.42.2.10 #define V 18.92

Corresponding van der Waals standard volume for our coordination number (cm^3/mol)

Definition at line 60 of file magpie.h.

6.42.2.11 #define Z 10.0

Surface coordination number used in the MSPD activity model.

Definition at line 52 of file magpie.h.

6.42.3 Function Documentation

6.42.3.1 double dq_dp (double *p*, const void * *data*, int *i*)

Function computes the derivative of the GSTA model with respect to partial pressure.

This function just computes the result of the derivative of GSTA isotherm model for the ith species at the given the partial pressure p.

Parameters

p	partial pressure in kPa at which to evaluate the GSTA model
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

6.42.3.2 double eMax (const void * *data*, int *i*)

Function to approximate the maximum lateral energy term for the ith species.

The function attempts to approximate the maximum lateral energy term for the ith species. This is not a true maximum, but a cheaper estimate. Value being computed is used to shift the geometric mean and formulate the average cross-lateral energy term between species i and j.

6.42.3.3 void eval_eta (const double * *par*, int *m_dat*, const void * *data*, double * *fvec*, int * *info*)

Function used with lmfit to evaluate the binary interaction parameters for each unique species pair.

This function is used to estimate the binary interaction parameters for all species pairs in a given sub-system. Those parameters are then stored for later used when evaluating the activity coefficients for the overall mixture. User does not need to call this function. GPAST will call automatically when needed.

Parameters

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the lmfit routine

6.42.3.4 void eval_GPAST (const double * *par*, int *m_dat*, const void * *data*, double * *fvec*, int * *info*)

Function used with lmfit to solve the GPAST system of equations.

This function is used after having calculated and stored all necessary information to solve a closed form GPAST

system of equations. User does not need to call this function. GPAST will call automatically when needed.

Parameters

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the lmfit routine

6.42.3.5 void eval_po (const double * *par*, int *m_dat*, const void * *data*, double * *fvec*, int * *info*)

Function used with lmfit to evaluate the reference state pressure of a species based on a sub-system.

This function is used to approximate reference state pressures based on the spreading pressure of a sub-system in GPAST. The sub-system will be one of the unique binary systems that exist in the overall mixed gas system. User does not need to call this function. GPAST will call automatically when needed.

Parameters

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the lmfit routine

6.42.3.6 void eval_po_PI (const double * *par*, int *m_dat*, const void * *data*, double * *fvec*, int * *info*)

Function used with lmfit to evaluate the reference state pressure of a species based on spreading pressure.

This function is used inside of the MSPD activity model to calculate the reference state pressure of a particular species at a given spreading pressure for the system. User does not need to call this function. GPAST will call automatically when needed.

Parameters

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the lmfit routine

6.42.3.7 void eval_po_qo (const double * *par*, int *m_dat*, const void * *data*, double * *fvec*, int * *info*)

Function used with lmfit to evaluate the reference state pressure of a species based on that species isotherm.

This function is used to evaluate the partial pressure or reference state pressure for a particular species given single-component adsorbed amount. User does not need to call this function. GPAST will call automatically when needed.

Parameters

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the lmfit routine

6.42.3.8 double grad_mSPD (const double * *par*, const void * *data*, int *i*)

Function to approximate the derivative of the MSPD activity model with spreading pressure.

This function returns a 2nd order, finite different approximation of the derivative of the MSPD activity model with the spreading pressure. The par argument will either hold the current iterates estimate of spreading pressure or should be passed as null. User does not need to call this function. GPAST will call automatically when needed.

Parameters

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure
i	ith species for which we will approximate the activty model gradient

6.42.3.9 void initialGuess_mSPD (double * *par*, const void * *data*)

Function to provide an initial guess to the unknown parameters being solved for in GPAST.

This function intends to provide an initial guess for the unknown values being solved for in the GPAST system. Depending on what type of solve is requested, this algorithm will provide a guess for the adsorbed or gas phase composition.

Parameters

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure

6.42.3.10 double lnact_mSPD (const double * *par*, const void * *data*, int *i*, volatile double *PI*)

Function to evaluate the MSPD activity coefficient for the ith species.

This function will return the natural log of the ith species activity coefficient using the Modified Spreading Pressure Dependent (MSPD) activity model. The par argument holds the variable values being solved for by GPAST and their contents will change depending on whether we are doing a forward or reverse evaluation. This function should not be called by the user and will only be called when needed in the GPAST routine.

Parameters

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure
i	ith species that we want to calculate the activity coefficient for
PI	lumped spreading pressure term used in gradient estimations

6.42.3.11 int MAGPIE (const void * *data*)

Function to call all sub-routines to solve a MAGPIE/GPAST problem at a given temperature and pressure.

This is the function that a typical user will want to incorporate into their own codes when evaluating adsorption of a gas mixture. Prior to calling this function, all required structures and information in the MAGPIE_DATA structure must have been properly initialized. After this function has completed it's operations, it will return an integer used to denote a success or failure of the routine. Integers 0, 1, 2, and 3 all denote success. Anything else is considered a failure.

To setup the MAGPIE_DATA structure correctly, you must reserve space for all vector objects based on the number of gas species in the mixture. In general, you only need to reserve space for the adsorbing species. However, you can also reserve space for non-adsorbing species, but you MUST give a gas/adsorbed mole fraction of the non-adsorbing species 0.0 so that the routine knows to ignore them (very important)!

After setting up the memory for the vector objects, you can intialize information specific to the simulation you want to request. The number of species (N), total pressure (PT) and gas temperature (T) must always be given. You can neglect the non-idealities of the surface phase by setting the Ideal bool to true. This will result in faster calculations,

because MAGPIE will just revert down to the Ideal Adsorbed Solution Theory (IAST).

The Recover bool will denote whether we are doing a forward or reverse GPAST evaluation. Forward evaluation is for solving for the composition of the adsorbed phase given the composition of the gas phase (Recover = false). Reverse evaluation is for solve for the composition of the gas phase given the composition of the adsorbed phase (Recover = true).

For a reverse evaluation (Recover = true) you will also need to stipulate whether or not there is a carrier gas (Carrier = true or false). A carrier gas is considered any non-adsorbing species that may be present in the gas phase and contributing to the total pressure in the system.

The parameters that must be initialized for all species include all GSTA_DATA parameters and the van der Waals volume parameter (v) in the mSPD_DATA structure. For non-adsorbing species, you can ignore these parameters, but need to set the sites (m) from GSTA_DATA to 1. GPAST cannot run any evaluations without these parameters being set properly AND set in the same order for all species (i.e., make sure that gpast_dat[i].qmax corresponds to mspd_dat[i].v and so on).

Lastly, you need to give either the gas phase or adsorbed phase mole fractions, depending on whether you are going to run a forward or reverse evaluation, respectively. For a forward evaluation, provide the gas mole fractions (y) in GPAST_DATA for each species (non-adsorbing species should have this value set to 0.0). For a reverse evaluation, provide the adsorbed mole fractions (x) in GPAST_DATA for each species, as well as the total adsorbed amount (qT) in SYSTEM_DATA. Again, non-adsorbing species should have their respective phase mole fractions set to 0.0 to exclude them from the simulation. Additionally, if there are non-adsorbing species present, then the Carrier bool in SYSTEM_DATA must be set to true.

Parameters

data	void pointer for the MAGPIE_DATA data structure holding all necessary information

6.42.3.12 double PI (double *po*, const void * *data*, int *i*)

Function computes the spreading pressure integral of the ith species.

This function uses an analytical solution to the spreading pressure integral with the GSTA isotherm to evaluate and return the value computed by that integral equation.

Parameters

po	partial pressure in kPa at which to evaluate the lumped spreading pressure
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

6.42.3.13 double q_p (double *p*, const void * *data*, int *i*)

Function computes the ratio between the adsorbed amount and partial pressure for the GSTA isotherm.

This function just computes the ratio between the adsorbed amount q (mol/kg) and the partial pressure p (kPa) at the given partial pressure. If p == 0, then this function returns the Henry's Law constant for the isotherm of the ith species.

Parameters

p	partial pressure in kPa at which to evaluate the GSTA model
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

6.42.3.14 double qo (double *po*, const void * *data*, int *i*)

Function computes the result of the GSTA isotherm for the ith species.

This function just computes the result of the GSTA isotherm model for the ith species given the partial pressure po.

Parameters

po	partial pressure in kPa at which to evaluate the GSTA model
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

Referenced by Matrix< T >::IntegralAvg(), and Matrix< T >::IntegralTotal().

6.42.3.15 double Qst (double *po*, const void * *data*, int *i*)

Function computes the heat of adsorption based on the ith species GSTA parameters.

This function computes the isosteric heat of adsorption (J/mol) for the GSTA parameters of the ith species.

Parameters

po	partial pressure in kPa at which to evaluate the heat of adsorption
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

6.42.3.16 double qT (const double * *par*, const void * *data*)

Function to calculate the total adsorbed amount (mol/kg) for the mixed surface phase.

This function will uses the obtained system parameters from par and estimate the total amount of gases adsorbed to the surface in mol/kg. The user does not need to call this function, since this result will be stored in the SYSTEM_DATA structure.

Parameters

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure

6.43 MAGPIE_Adsorption.h File Reference

Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system.

```
#include "AuxKernel.h"
#include "flock.h"
```

Classes

- class MAGPIE_Adsorption

  Magpie Adsorption class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< MAGPIE_Adsorption > ()

6.43.1 Detailed Description

Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system. This file is responsible for calculating the adsorption equilibria of a particular species in the system. The MAGPIE object is stored as a material property whose constants are set in the corresponding material property file (see MagpieAdsorbateProperties.h). That information is then used to call the MAGPIE routine to calculate the mixed gas adsorption for a specific species of interest.

Unfortunately, the material property system has recently changed in MOOSE, making this operation much less efficient. Under the new system, all material properties are declared as constants when outside of their respective material property files. This means that in order for me to call the MAGPIE subroutine, which edits values in the MAGPIE object, I have to create a copy of the entire object and have the subroutine act on that copy.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file MAGPIE_Adsorption.h.

6.43.2 Function Documentation

6.43.2.1 template<> InputParameters validParams< MAGPIE_Adsorption > ( )

6.44 MAGPIE_AdsorptionHeat.h File Reference

Auxillary kernel to calculate heat of adsorption of a particular gas species in the system.

```
#include "AuxKernel.h"
#include "flock.h"
```

Classes

- class MAGPIE_AdsorptionHeat
  Magpie Adsorption Heat class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< MAGPIE_AdsorptionHeat > ()

6.44.1 Detailed Description

Auxillary kernel to calculate heat of adsorption of a particular gas species in the system. This file is responsible for calculating the heat of adsorption of a particular species in the system. The MAGPIE object is stored as a material property whose constants are set in the corresponding material property file (see MagpieAdsorbateProperties.h). That information is then used to call the MAGPIE routine to calculate the mixed gas adsorption for a specific species of interest.

Unfortunately, the material property system has recently changed in MOOSE, making this operation much less efficient. Under the new system, all material properties are declared as constants when outside of their respective material property files. This means that in order for me to call the MAGPIE subroutine, which edits values in the MAGPIE object, I have to create a copy of the entire object and have the subroutine act on that copy.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MAGPIE_AdsorptionHeat.h.

6.44.2 Function Documentation

6.44.2.1 template<> InputParameters validParams< MAGPIE_AdsorptionHeat > ( )

6.45 MAGPIE_Perturbation.h File Reference

Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system.

```
#include "AuxKernel.h"
#include "flock.h"
```

Classes

- class MAGPIE_Perturbation

  Magpie Perturbation class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< MAGPIE_Perturbation > ()

6.45.1 Detailed Description

Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system. This file is responsible for calculating the perturbed adsorption equilibria of a particular species in the system. The MAGPIE object is stored as a material property whose constants are set in the corresponding material property file (see MagpieAdsorbateProperties.h). That information is then used to call the MAGPIE routine to calculate the mixed gas perturbed adsorption for a specific species of interest.

The perturbation is used to approximate the strength of adsorption via first order finite difference. That adsorption strength is then loosely coupled to the gaseous species non-linear variable through a retardation coefficient in the mass transport equations. We use loose coupling to improve the efficiency of the solutions for this multi-scale mass transfer problem. Full coupling would result in significant losses in efficiency, or even complete failure to converge. DO NOT TRY FULL COUPLING!

Unfortunately, the material property system has recently changed in MOOSE, making this operation much less efficient. Under the new system, all material properties are declared as constants when outside of their respective

material property files. This means that in order for me to call the MAGPIE subroutine, which edits values in the MAGPIE object, I have to create a copy of the entire object and have the subroutine act on that copy.

Note

We will only use this kernel to approximate the retardation effect of adsorption IF we are neglecting the micro-scale kinetics of adsorption/mass transfer into the adsorbent pellets. Kinetics coupling will be accomplished in another kernel.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MAGPIE_Perturbation.h.

6.45.2 Function Documentation

6.45.2.1 template<> InputParameters validParams< MAGPIE_Perturbation >( )

6.46 MagpieAdsorbateProperties.h File Reference

Material Properties kernel that will setup and hold all information associated with MAGPIE simulations.

```
#include "Material.h"
#include "flock.h"
```

Classes

- class MagpieAdsorbateProperties

  MagpieAdsorbateProperties class object inherits from Material object.

Functions

- template<>
  InputParameters validParams< MagpieAdsorbateProperties > ()

6.46.1 Detailed Description

Material Properties kernel that will setup and hold all information associated with MAGPIE simulations. This file creates a material property object for the MAGPIE data structure and associated constants. That information is used in conjunction with the MAGPIE simulation functions (see magpie.h) in order to approximate the adsorption capacities and adsorbed amounts of each gas species in a given system for a given adsorbent.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MagpieAdsorbateProperties.h.

6.46.2 Function Documentation

6.46.2.1 template<> InputParameters validParams< MagpieAdsorbateProperties >( )

6.47 scopsowl.h File Reference

Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems.

```
#include "egret.h"
#include "skua.h"
```

Classes

- struct SCOPSOWL_PARAM_DATA

  Data structure for the species' parameters in SCOPSOWL.
- struct SCOPSOWL_DATA

  Primary data structure for SCOPSOWL simulations.

Macros

- #define SCOPSOWL_HPP_
- #define Dp(Dm, ep) (ep∗ep∗Dm)

  Estimate of Pore Diffusivity (cm^2/s)
- #define Dk(rp, T, MW) (9700.0∗rp∗pow((T/MW),0.5))

  Estimate of Knudsen Diffusivity (cm^2/s)
- #define avgDp(Dp, Dk) (pow(((1/Dp)+(1/Dk)),-1.0))

  Estimate of Average Pore Diffusion (cm^2/s)

Functions

- void print2file_species_header (FILE ∗Output, SCOPSOWL_DATA ∗owl_dat, int i)

  Function to print out the main header for the output file.
- void print2file_SCOPSOWL_time_header (FILE ∗Output, SCOPSOWL_DATA ∗owl_dat, int i)

  Function to print out the time and space header for the output file.

- void print2file_SCOPSOWL_header (SCOPSOWL_DATA *owl_dat)
  Function to call the species and time header functions.
- void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA *owl_dat)
  Function to print out the old time results to the output file.
- void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA *owl_dat)
  Function to print out the new time results to the output file.
- double default_adsorption (int i, int l, const void *user_data)
  Default function for evaluating adsorption and adsorption strength.
- double default_retardation (int i, int l, const void *user_data)
  Default function for evaluating retardation coefficient.
- double default_pore_diffusion (int i, int l, const void *user_data)
  Default function for evaluating pore diffusivity.
- double default_surf_diffusion (int i, int l, const void *user_data)
  Default function for evaluating surface diffusion for HOMOGENEOUS pellets.
- double default_effective_diffusion (int i, int l, const void *user_data)
  Default function for evaluating effective diffusivity for HOMOGENEOUS pellets.
- double const_pore_diffusion (int i, int l, const void *user_data)
  Constant pore diffusion function for homogeneous or heterogeneous pellets.
- double default_filmMassTransfer (int i, const void *user_data)
  Default function for evaluating the film mass transfer coefficient.
- double const_filmMassTransfer (int i, const void *user_data)
  Constant film mass transfer coefficient function.
- int setup_SCOPSOWL_DATA (FILE *file, double(*eval_sorption)(int i, int l, const void *user_data), double(*eval_retardation)(int i, int l, const void *user_data), double(*eval_pore_diff)(int i, int l, const void *user_data), double(*eval_filmMT)(int i, const void *user_data), double(*eval_surface_diff)(int i, int l, const void *user_data), const void *user_data, MIXED_GAS *gas_data, SCOPSOWL_DATA *owl_data)
  Setup function to allocate memory and setup function pointers for the SCOPSOWL simulation.
- int SCOPSOWL_Executioner (SCOPSOWL_DATA *owl_dat)
  SCOPSOWL executioner function to solve a time step.
- int set_SCOPSOWL_ICs (SCOPSOWL_DATA *owl_dat)
  Function to set the initial conditions for a SCOPSOWL simulation.
- int set_SCOPSOWL_timestep (SCOPSOWL_DATA *owl_dat)
  Function to set the timestep of the SCOPSOWL simulation.
- int SCOPSOWL_preprocesses (SCOPSOWL_DATA *owl_dat)
  Function to perform all preprocess SCOPSOWL operations.
- int set_SCOPSOWL_params (const void *user_data)
  Function to set the values of all non-linear system parameters during simulation.
- int SCOPSOWL_postprocesses (SCOPSOWL_DATA *owl_dat)
  Function to perform all postprocess SCOPSOWL operations.
- int SCOPSOWL_reset (SCOPSOWL_DATA *owl_dat)
  Function to reset all stateful information to prepare for next simulation.
- int SCOPSOWL (SCOPSOWL_DATA *owl_dat)
  Function to progress the SCOPSOWL simulation through time till complete.

6.47.1 Detailed Description

Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems. scopsowl.cpp

This file contains structures and functions associated with modeling adsorption in commercial, bi-porous adsorbents such as zeolites and mordenites. The pore diffusion and mass transfer equations are coupled with adsorption and surface diffusion through smaller crystals embedded in a binder matrix. However, you can also direct this simulation to treat the adsorbent as homogeneous (instead of heterogeneous) in order to model an even greater variety of gaseous adsorption kinetic problems. This object is coupled with either MAGPIE, SKUA, or BOTH depending on the type of simulation requested.

Author

Austin Ladshaw

Date

01/29/2015

Copyright

Definition in file scopsowl.h.

6.47.2 Macro Definition Documentation

6.47.2.1 #define avgDp(Dp, Dk) (pow(((1/Dp)+(1/Dk)),-1.0))

Estimate of Average Pore Diffusion (cm^2/s)

Definition at line 37 of file scopsowl.h.

6.47.2.2 #define Dk(rp, T, MW) (9700.0∗rp∗pow((T/MW),0.5))

Estimate of Knudsen Diffusivity (cm^2/s)

Definition at line 33 of file scopsowl.h.

6.47.2.3 #define Dp(Dm, ep) (ep∗ep∗Dm)

Estimate of Pore Diffusivity (cm^2/s)

Definition at line 29 of file scopsowl.h.

6.47.2.4 #define SCOPSOWL_HPP_

Definition at line 26 of file scopsowl.h.

6.47.3 Function Documentation

6.47.3.1 double const_filmMassTransfer (int *i*, const void ∗ *user_data*)

Constant film mass transfer coefficient function.

This function is used when the user wants to specify a constant value for film mass transfer. The value of that coefficient is then set equal to the value of film_transfer in the SCOPSOWL_PARAM_DATA structure.

Parameters

i	index for the ith species in the system
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.2 double const_pore_diffusion (int *i*, int *l*, const void ∗ *user_data*)

Constant pore diffusion function for homogeneous or heterogeneous pellets.

This function should be used if the user wants to specify a constant pore diffusivity. The value of pore diffusion is then set equal to the value of pore_diffusion in the SCOPSOWL_PARAM_DATA structure.

Parameters

i	index for the ith species in the system
l	index for the lth node in the macro-scale domain
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.3 double default_adsorption (int *i*, int *l*, const void * *user_data*)

Default function for evaluating adsorption and adsorption strength.

This function is called in the preprocesses and postprocesses to estimate the strength of adsorption in the macro-scale problem from perturbations. It will use perturbations in either the MAGPIE simulation or SKUA simulation, depending on the type of problem the user is solving.

Parameters

i	index for the ith species in the system
l	index for the lth node in the macro-scale domain
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.4 double default_effective_diffusion (int *i*, int *l*, const void * *user_data*)

Default function for evaluating effective diffusivity for HOMOGENEOUS pellets.

This function is ONLY used if the pellet is determined to be homogeneous. Otherwise, this is replaced by the pore diffusion function. The effective diffusivity is determined by the combination of pore diffusivity and surface diffusivity with adsorption strength in an homogeneous pellet.

Parameters

i	index for the ith species in the system
l	index for the lth node in the macro-scale domain
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.5 double default_filmMassTransfer (int *i*, const void * *user_data*)

Default function for evaluating the film mass transfer coefficient.

This function is called during the setup of the boundary conditions and is used to estimate the film mass transfer coefficient for the macro-scale problem. The coefficient is calculated according to the kinetic theory of gases (see egret.h).

Parameters

i	index for the ith species in the system
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.6 double default_pore_diffusion (int *i*, int *l*, const void * *user_data*)

Default function for evaluating pore diffusivity.

This function is called during the evaluation of non-linear residuals to more accurately represent non-linearities in the pore diffusion behavior. The pore diffusion is calculated based on kinetic theory of gases (see egret.h) and is adjusted according to the Knudsen Diffusion model and the porosity of the binder material.

Parameters

i	index for the ith species in the system
l	index for the lth node in the macro-scale domain
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.7 double default_retardation (int *i*, int *l*, const void * *user_data*)

Default function for evaluating retardation coefficient.

This function is called in the preprocesses and postprocesses to estimate the retardation coefficient for the simulation. It is recalculated at every time step to keep track of all changing conditions in the simulation.

Parameters

i	index for the ith species in the system
l	index for the lth node in the macro-scale domain
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.8 double default_surf_diffusion (int *i*, int *l*, const void * *user_data*)

Default function for evaluating surface diffusion for HOMOGENEOUS pellets.

This function is ONLY used if the pellet is determined to be homogeneous. Otherwise, this is replaced by the surface diffusion function for the SKUA simulation. The diffusivity is calculated based on the Arrhenius rate expression and then adjusted by the outside partial pressure of the adsorbing species.

Parameters

i	index for the ith species in the system
l	index for the lth node in the macro-scale domain
user_data	pointer for the SCOSPOWL_DATA structure

6.47.3.9 void print2file_SCOPSOWL_header (SCOPSOWL_DATA * *owl_dat*)

Function to call the species and time header functions.

6.47.3.10 void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA * *owl_dat*)

Function to print out the new time results to the output file.

6.47.3.11 void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA * *owl_dat*)

Function to print out the old time results to the output file.

6.47.3.12 void print2file_SCOPSOWL_time_header (FILE * *Output*, SCOPSOWL_DATA * *owl_dat*, int *i*)

Function to print out the time and space header for the output file.

6.47.3.13 void print2file_species_header (FILE * *Output*, SCOPSOWL_DATA * *owl_dat*, int *i*)

Function to print out the main header for the output file.

6.47.3.14 int SCOPSOWL (SCOPSOWL_DATA * *owl_dat*)

Function to progress the SCOPSOWL simulation through time till complete.

This function will call the initial conditions, then progressively call the executioner, time step, and reset functions to propagate the simulation in time. As such, this function is primarily used when running a SCOPSOWL simulation by itself and not when coupling it to an other problem.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.15 int SCOPSOWL_Executioner (SCOPSOWL_DATA * owl_dat)

SCOPSOWL executioner function to solve a time step.

This function will call the preprocess, solver, and postprocess functions to evaluate a single time step in a simulation. All simulation conditions must be set prior to calling this function. This function will typically be the one called from other simulations that will involve a SCOPSOWL evaluation to resolve kinetic coupling.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.16 int SCOPSOWL_postprocesses (SCOPSOWL_DATA * owl_dat)

Function to perform all postprocess SCOPSOWL operations.

This function will update the retardation coefficients based on newly obtained simulation results for the current time step and calculate the average and total amount of adsorption of each species in the domain. Additionally, this function will call the print functions to store simulation results in the output file.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.17 int SCOPSOWL_preprocesses (SCOPSOWL_DATA * owl_dat)

Function to perform all preprocess SCOPSOWL operations.

This function will update the boundary conditions and simulation conditions based on the current temperature, pressure, and gas phase composition, which may all vary in time. Since this function is called by the SCOPSOWL-_Executioner, it does not need to be called explicitly by the user.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.18 int SCOPSOWL_reset (SCOPSOWL_DATA * owl_dat)

Function to reset all stateful information to prepare for next simulation.

This function will update the stateful information used in SCOPSOWL to prepare the system for the next time step in the simulation. However, because updating the states erases the old state, the user must be absolutely sure that the simulation is ready to be updated. For just running standard simulations, this is not an issue, but in coupling with other simulations it is very important.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.19 int set_SCOPSOWL_ICs (SCOPSOWL_DATA * owl_dat)

Function to set the initial conditions for a SCOPSOWL simulation.

This function will setup the initial conditions of the simulation based on the initial temperature, pressure, and adsorbed molefractions. It assumes that the initial conditions are constant throughout the domain of the problem. This function should only be called once during a simulation.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.20 int set_SCOPSOWL_params (const void * *user_data*)

Function to set the values of all non-linear system parameters during simulation.

This is the function override for the FINCH setparams function (see finch.h). It will update the values of non-linear parameters in the residuals so that all variables in a species' system are fully coupled.

Parameters

user_data	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.21 int set_SCOPSOWL_timestep (SCOPSOWL_DATA * *owl_dat*)

Function to set the timestep of the SCOPSOWL simulation.

This function is used to set the next time step to be used in the SCOPSOWL simulation. A constant time step based on the size of the pellet discretization will be used. Users may want to use a custom time step to ensure that coupled-multi-scale systems are all in sync.

Parameters

owl_dat	pointer to the SCOPSOWL_DATA structure (must be initialized)

6.47.3.22 int setup_SCOPSOWL_DATA (FILE * *file,* double(*)(int i, int l, const void *user_data) *eval_sorption,* double(*)(int i, int l, const void *user_data) *eval_retardation,* double(*)(int i, int l, const void *user_data) *eval_pore_diff,* double(*)(int i, const void *user_data) *eval_filmMT,* double(*)(int i, int l, const void *user_data) *eval_surface_diff,* const void * *user_data,* MIXED_GAS * *gas_data,* SCOPSOWL_DATA * *owl_data*)

Setup function to allocate memory and setup function pointers for the SCOPSOWL simulation.

This function sets up the memory and function pointers used in SCOPSOWL simulations. User can provide NULL in place of functions for the function pointers and the setup will automatically use just the default settings. However, the user is required to pass the necessary data structure pointers for MIXED_GAS and SCOPSOWL_DATA.

Parameters

file	pointer to the output file to print out results
eval_sorption	pointer to the adsorption evaluation function
eval_retardation	pointer to the retardation evaluation function
eval_pore_diff	pointer to the pore diffusion function
eval_filmMT	pointer to the film mass transfer function
eval_surface_diff	pointer to the surface diffusion function (required)
user_data	pointer to the user's data structure used for the parameter functions
gas_data	pointer to the MIXED_GAS structure used to evaluate kinetic gas theory
owl_data	pointer to the SCOPSOWL_DATA structure

6.48 skua.h File Reference

Surface Kinetics for Uptake by Adsorption.

```
#include "finch.h"
#include "magpie.h"
#include "egret.h"
```

Classes

- struct SKUA_PARAM

  Data structure for species' parameters in SKUA.

- struct SKUA_DATA

  Data structure for all simulation information in SKUA.

Macros

- #define SKUA_HPP_
- #define D_inf(Dref, Tref, B, p, T) (Dref ∗ pow(p+sqrt(DBL_EPSILON),(Tref/T)-B))

  Empirical correction of diffusivity (um^2/hr)
- #define D_o(Diff, E, T) (Diff ∗ exp(-E/(Rstd∗T)))

  Arrhenius Rate Expression for Diffusivity (um^2/hr)
- #define D_c(Diff, phi) (Diff ∗ (1.0/((1.0+1.1E-6)-phi)))

  Approximate Darken Diffusivity Equation (um^2/hr)

Functions

- void print2file_species_header (FILE ∗Output, SKUA_DATA ∗skua_dat, int i)

  Function to print out the species' headers to output file.
- void print2file_SKUA_time_header (FILE ∗Output, SKUA_DATA ∗skua_dat, int i)

  Function to print out time and space headers to output file.
- void print2file_SKUA_header (SKUA_DATA ∗skua_dat)

  Function calls the other header functions to establish output file structure.
- void print2file_SKUA_results_old (SKUA_DATA ∗skua_dat)

  Function to print out the old time step simulation results to the output file.
- void print2file_SKUA_results_new (SKUA_DATA ∗skua_dat)

  Function to print out the new time step simulation results to the output file.
- double default_Dc (int i, int l, const void ∗data)

  Default function for surface diffusivity.
- double default_kf (int i, const void ∗data)

  Default function for film mass transfer coefficent.
- double const_Dc (int i, int l, const void ∗data)

  Constant surface diffusivity function.
- double simple_darken_Dc (int i, int l, const void ∗data)

  Simple Darken model for surface diffusivity.
- double theoretical_darken_Dc (int i, int l, const void ∗data)

  Theoretical Darken model for surface diffusivity.
- double empirical_kf (int i, const void ∗data)

  Empirical function for film mass transfer coefficent.
- double const_kf (int i, const void ∗data)

  Constant function for film mass transfer coefficent.
- int molefractionCheck (SKUA_DATA ∗skua_dat)

  Function to check mole fractions in gas and solid phases for errors.
- int setup_SKUA_DATA (FILE ∗file, double(∗eval_Dc)(int i, int l, const void ∗user_data), double(∗eval_Kf)(int i, const void ∗user_data), const void ∗user_data, MIXED_GAS ∗gas_data, SKUA_DATA ∗skua_dat)

  Function to setup the function pointers and vector objects in memory to setup the SKUA simulation.
- int SKUA_Executioner (SKUA_DATA ∗skua_dat)

  Function to execute preprocesses, solvers, and postprocesses for a SKUA simulation.
- int set_SKUA_ICs (SKUA_DATA ∗skua_dat)

  Function to establish the initial conditions of adsorption in the adsorbent.
- int set_SKUA_timestep (SKUA_DATA ∗skua_dat)

  Function to establish the time step for the current simulation.
- int SKUA_preprocesses (SKUA_DATA ∗skua_dat)

Function to perform the necessary preprocess operations before a solve.

- int set_SKUA_params (const void *user_data)

  Function to call the diffusivity function during the solve.

- int SKUA_postprocesses (SKUA_DATA *skua_dat)

  Function to perform the necessary postprocess operations after a solve.

- int SKUA_reset (SKUA_DATA *skua_dat)

  Function to reset the stateful information in SKUA after a simulation.

- int SKUA (SKUA_DATA *skua_dat)

  Function to iteratively call all execution steps to evolve a simulation through time.

6.48.1 Detailed Description

Surface Kinetics for Uptake by Adsorption. skua.cpp

This file contains structures and functions associated with solving the surface diffusion partial differential equations for adsorption kinetics in spherical and/or cylindrical adsorbents. For this system, it is assumed that the pore size is so small that all molecules are confined to movement exclusively on the surface area of the adsorbent. The total amount of adsorption for each species is drive by the MAGPIE model for non-ideal mixed gas adsorption. Spatial and temporal varience in adsorption is caused by a combination of different kinetics between adsorbing species and different adsorption affinities for the surface.

The function for surface diffusion involves four parameters, although not all of these parameters are required to be used. Surface diffusion theoretically varies with temperature according to the Arrhenius rate expression, but we also add in an empirical correction term to account for variations in diffusivity with the partial pressure of the species in the gas phase.

D_surf = D_ref * exp(-E / (R*T)) * pow(p , (T_ref/T) - B)

D_ref is the Reference Diffusivity (um^2/hr), E is the activation energy for adsorption (J/mol), R is the gas law constant (J/K/mol), T is the system temperature (K), p is the partial pressure of the adsorbing species (kPa), T_ref is the Reference Temperature (K), and B is the Affinity constant.

Author

Austin Ladshaw

Date

01/26/2015

Copyright

Definition in file skua.h.

6.48.2 Macro Definition Documentation

6.48.2.1 #define D_c(*Diff, phi*) (Diff * (1.0/((1.0+1.1E-6)-phi)))

Approximate Darken Diffusivity Equation (um^2/hr)

Definition at line 48 of file skua.h.

6.48.2.2 #define D_inf(*Dref, Tref, B, p, T*) (Dref * pow(p+sqrt(DBL_EPSILON),(Tref/T)-B))

Empirical correction of diffusivity (um^2/hr)

Definition at line 40 of file skua.h.

6.48.2.3 #define D_o(*Diff, E, T*) (Diff ∗ exp(-E/(Rstd∗T)))

Arrhenius Rate Expression for Diffusivity (um^2/hr)

Definition at line 44 of file skua.h.

6.48.2.4 #define SKUA_HPP_

Definition at line 37 of file skua.h.

6.48.3 Function Documentation

6.48.3.1 double const_Dc (int *i*, int *l*, const void ∗ *data*)

Constant surface diffusivity function.

This function allows the user to specify just a single constant value for surface diffusivity. The value of diffusivity applied at all nodes will be the ref_diffusion parameter in SKUA_PARAM.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
l	index of the node in the spatial discretization that this function acts on
data	pointer to the SKUA_DATA structure

6.48.3.2 double const_kf (int *i*, const void ∗ *data*)

Constant function for film mass transfer coefficent.

This function allows the user to specify a constant value for the film mass transfer coefficient. The value of the film mass transfer coefficient will be the value of film_transfer given in the SKUA_PARAM data structure.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
data	pointer to the SKUA_DATA structure

6.48.3.3 double default_Dc (int *i*, int *l*, const void ∗ *data*)

Default function for surface diffusivity.

This is the default function provided by SKUA for the calculation of the surface diffusivity parameter. The diffusivity is calculated based on the Arrhenius rate expression, then corrected for using the empirical correction term with the outside partial pressure of the gas species.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
l	index of the node in the spatial discretization that this function acts on
data	pointer to the SKUA_DATA structure

6.48.3.4 double default_kf (int *i*, const void ∗ *data*)

Default function for film mass transfer coefficent.

This is the default function provided by SKUA for the calculation of the film mass transfer parameter. By default, we are usually going to couple the SKUA model with a pore diffusion model (see scopsowl.h). Therefore, the film mass transfer coefficient would be zero, because we would only consider a Dirichlet boundary condition for this sub-problem.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
data	pointer to the SKUA_DATA structure

6.48.3.5 double empirical_kf (int *i*, const void * *data*)

Empirical function for film mass transfer coefficent.

This function provides an empirical estimate of the mass transfer coefficient using the gas velocity, molecular diffusivities, and dimensionless numbers (see egret.h). It is used as the default film mass transfer function IF the boundary condition is specified to be a Neumann type boundary by the user.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
data	pointer to the SKUA_DATA structure

6.48.3.6 int molefractionCheck (SKUA_DATA * *skua_dat*)

Function to check mole fractions in gas and solid phases for errors.

This function is called after reading input and before calling the primary solution routines. It will force and error and quit the program if their are inconsistencies in the mole fractions it was given. All mole fractions must sum to 1, otherwise there is missing information.

6.48.3.7 void print2file_SKUA_header (SKUA_DATA * *skua_dat*)

Function calls the other header functions to establish output file structure.

6.48.3.8 void print2file_SKUA_results_new (SKUA_DATA * *skua_dat*)

Function to print out the new time step simulation results to the output file.

6.48.3.9 void print2file_SKUA_results_old (SKUA_DATA * *skua_dat*)

Function to print out the old time step simulation results to the output file.

6.48.3.10 void print2file_SKUA_time_header (FILE * *Output*, SKUA_DATA * *skua_dat*, int *i*)

Function to print out time and space headers to output file.

6.48.3.11 void print2file_species_header (FILE * *Output*, SKUA_DATA * *skua_dat*, int *i*)

Function to print out the species' headers to output file.

6.48.3.12 int set_SKUA_ICs (SKUA_DATA * *skua_dat*)

Function to establish the initial conditions of adsorption in the adsorbent.

This function needs to be called before doing any simulation or execution of a time step, but only once per simulation. It sets the value of adsorption for each adsorbable species to the specified initial values given via qT and xIC in SKUA_DATA.

6.48.3.13 int set_SKUA_params (const void * *user_data*)

Function to call the diffusivity function during the solve.

This is the function passed into FINCH to be called during the FINCH solver (see finch.h). It will call the diffusion functions set by the user in the setup function above. This is not overridable.

6.48.3.14 int set_SKUA_timestep (SKUA_DATA * *skua_dat*)

Function to establish the time step for the current simulation.

This function is called to set a time step value for a particular simulation step. By default, the time step is set to (1/4)x space step size. If you need to change the step size, you must do so manually.

6.48.3.15 int setup_SKUA_DATA (FILE * *file*, double(*)(int i, int l, const void *user_data) *eval_Dc*, double(*)(int i, const void *user_data) *eval_Kf*, const void * *user_data*, MIXED_GAS * *gas_data*, SKUA_DATA * *skua_dat*)

Function to setup the function pointers and vector objects in memory to setup the SKUA simulation.

This function is called to setup the SKUA problem in memory and set function pointers to either defaults or user specified functions. It must be called prior to calling any other SKUA function and will report an error if the object was not setup properly.

Parameters

file	pointer to the output file for SKUA simulations
eval_Dc	pointer to the function to evaluate the surface diffusivity
eval_Kf	pointer to the function to evaluate the film mass transfer coefficient
user_data	pointer to a user defined data structure used in the calculation the the parameters
gas_data	pointer to the MIXED_GAS data structure for egret.h calculations
skua_dat	pointer to the SKUA_DATA data structure

6.48.3.16 double simple_darken_Dc (int *i*, int *l*, const void * *data*)

Simple Darken model for surface diffusivity.

This function uses an approximation to Darken's model for surface diffusion. The approximation is exact if the isotherm for adsorption takes the form of the Langmuir model, but is only approximate if the isotherm is heterogeneous. Forming the approximation in this manner is significantly cheaper than forming the true Darken model expression for the GSTA isotherm.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
l	index of the node in the spatial discretization that this function acts on
data	pointer to the SKUA_DATA structure

6.48.3.17 int SKUA (SKUA_DATA * *skua_dat*)

Function to iteratively call all execution steps to evolve a simulation through time.

This function is used in conjunction with the scenario call from the UI to numerically solve the adsorption kinetics problem in time. It will call the initial conditions function once, then iteratively call the reset, time step, and executioner functions for SKUA to push the simulation forward in time. This function will be called from the SKUA_SCENARIOS function.

6.48.3.18 int SKUA_Executioner (SKUA_DATA * *skua_dat*)

Function to execute preprocesses, solvers, and postprocesses for a SKUA simulation.

This function calls the preprocess, solver, and postprocess functions to complete a single time step in a SKUA simulation. User's will want to call this function whenever a time step simulation result is needed. This is used primarily when coupling with other models (see scopsowl.h).

6.48.3.19 int SKUA_postprocesses (SKUA_DATA * *skua_dat*)

Function to perform the necessary postprocess operations after a solve.

This function performs postprocess operations after a solve was completed successfully. Those operations include

estimating average total adsorption, average adsorbed mole fractions, and heat of adsorption for each species. Results are then printed to the output file.

6.48.3.20 int SKUA_preprocesses (SKUA_DATA * *skua_dat*)

Function to perform the necessary preprocess operations before a solve.

This function performs preprocess operations prior to calling the solver routine. Those preprocesses include establishing boundary conditions and performing a MAGPIE simulation for the adsorption on the surface (see magpie.h).

6.48.3.21 int SKUA_reset (SKUA_DATA * *skua_dat*)

Function to reset the stateful information in SKUA after a simulation.

This function sets all the old state data to the newly formed state data. It needs to be called after a successful execution of the simulation step and before calling for the next time step to be solved. Do not call out of turn, otherwise information will be lost.

6.48.3.22 double theoretical_darken_Dc (int *i*, int *l*, const void * *data*)

Theoretical Darken model for surface diffusivity.

This function uses the full theoretical expression of the Darken's diffusion model to calculate the surface diffusivity. This calculation involves formulating the reference state pressures for the adsorbed amount at every node, then calculating derivatives of the adsorption isotherm for each species. It is more accurate than the simple Darken model function, but costs significantly more computational time.

Parameters

i	index of the gas/adsorbed phase species that this function acts on
l	index of the node in the spatial discretization that this function acts on
data	pointer to the SKUA_DATA structure

6.49 TotalColumnPressure.h File Reference

Auxillary kernel to calculate total column pressure based on temperature and concentrations.

```
#include "AuxKernel.h"
```

Classes

- class TotalColumnPressure

  Total Column Pressure class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< TotalColumnPressure > ()

6.49.1 Detailed Description

Auxillary kernel to calculate total column pressure based on temperature and concentrations. This file is responsible for calculating the total column pressure in a fixed-bed adsorber given the temperature and the concentrations of all species in the gas phase. The gas phase is assumed to behave ideally and ideal gas law is employed to estimate the pressure.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owernship or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file TotalColumnPressure.h.

6.49.2 Function Documentation

6.49.2.1 template<> InputParameters validParams< TotalColumnPressure >( )

6.50 TotalPressureIC.h File Reference

Initial Condition kernel for initial temperature in a fixed-bed column.

```
#include "InitialCondition.h"
```

Classes

- class TotalPressureIC

  TotalPressureIC class object inherits from InitialCondition object.

Functions

- template<>
  InputParameters validParams< TotalPressureIC > ()

6.50.1 Detailed Description

Initial Condition kernel for initial temperature in a fixed-bed column. This file creates an initial condition for the temperature in the bed. The initial condition for temperature is assumed a constant value at all points in the bed. However, this can be modified later to include spatially varying initial conditions for temperature.

Note

If you want to have spatially varying initial conditions, you will need to modify the virtual value function of this kernel. Otherwise, it is assumed that the non-linear variable is initially constant at all points in the domain.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file TotalPressureIC.h.

6.50.2 Function Documentation

6.50.2.1 template<> InputParameters validParams< TotalPressureIC >( )

6.51 WallAmbientHeatTransfer.h File Reference

Standard kernel for the transfer of heat from the column wall to the ambient air.

```
#include "Kernel.h"
```

Classes

- class WallAmbientHeatTransfer

  WallAmbientHeatTransfer class object inherits from Kernel object.

Functions

- template<>
  InputParameters validParams< WallAmbientHeatTransfer > ()

6.51.1 Detailed Description

Standard kernel for the transfer of heat from the column wall to the ambient air. This file creates a standard MOOSE kernel for the transfer of energy as heat between the walls of the column and the ambient air or some radient outer heat source/sink. The heat transfer is based on the thickness of the wall and a bed-wall heat transfer coefficient. It is coupled to the ambient heat and is a primary kernel used in determining the heat of the wall.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file WallAmbientHeatTransfer.h.

6.51.2 Function Documentation

6.51.2.1 template<> InputParameters validParams< WallAmbientHeatTransfer > ( )

6.52 WallHeatAccumulation.h File Reference

Time Derivative kernel for the accumulation of heat in a walls of the column.

```
#include "TimeDerivative.h"
```

Classes

- class WallHeatAccumulation

  WallHeatAccumulation class object inherits from TimeDerivative object.

Functions

- template<>
  InputParameters validParams< WallHeatAccumulation > ()

6.52.1 Detailed Description

Time Derivative kernel for the accumulation of heat in a walls of the column. This file creates a time derivative kernel to be used in the energy balance equations for accumulation of heat in the column wall. It combines the retardation coefficient from a material property with the standard time derivative kernel object in MOOSE.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any ownership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file WallHeatAccumulation.h.

6.52.2 Function Documentation

6.52.2.1 template<> InputParameters validParams< WallHeatAccumulation > ( )

Index

~DgospreyApp
DgospreyApp, 82
~Matrix
Matrix, 146
_Diffusion
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 48
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
DGFluxBC, 62
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 75
DGMassFluxLimitedBC, 79
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 113
_Dxx
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 48
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
DGFluxBC, 62
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 75
DGMassFluxLimitedBC, 79
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 113
_Dxy
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 48
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
DGFluxBC, 62
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 79
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 113
_Dxz
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 48
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 62
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 79
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 113
_Dyx
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 48
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 79
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 113
_Dyy
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 65
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 114
_Dyz
DGAnisotropicDiffusion, 43
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 65
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 114
_Dzx

DGAnisotropicDiffusion, 44
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 54
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 65
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 114
_Dzy
DGAnisotropicDiffusion, 44
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 54
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 65
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
GAnisotropicDiffusion, 104
GColumnHeatDispersion, 109
GColumnMassDispersion, 114
_Dzz
DGAnisotropicDiffusion, 44
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 54
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 66
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 72
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
GAnisotropicDiffusion, 105
GColumnHeatDispersion, 109
GColumnMassDispersion, 114
_Kz
BedProperties, 25
_PT_IC
ConcentrationIC, 37
TotalPressureIC, 190
_TC_IC
ColumnTemperatureIC, 36
_T_IC
ConcentrationIC, 37
_Ua
BedProperties, 25
_Uw
BedProperties, 25
_ambient_temp
WallAmbientHeatTransfer, 192
_bed_wall_transfer_coeff
BedProperties, 24
BedWallHeatTransfer, 27
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
_binder_fraction
AdsorbentProperties, 11
_binder_porosity
AdsorbentProperties, 11
_coef
CoupledLDF, 39
LinearDrivingForce, 134
_column_length
FlowProperties, 98
_column_temp
BedWallHeatTransfer, 27
_comp_Sutherland_const
FlowProperties, 99
_comp_heat_capacity
FlowProperties, 98
_comp_ref_temp
FlowProperties, 98
_comp_ref_viscosity
FlowProperties, 98
_conductivity
BedProperties, 24
DGColumnHeatDispersion, 48
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 72
GColumnHeatDispersion, 109
_crystal_radius
AdsorbentProperties, 11
_din
BedProperties, 24
_dispersion
DGColumnMassDispersion, 53
DGMassFluxBC, 75
DGMassFluxLimitedBC, 79
FlowProperties, 99
GColumnMassDispersion, 113
_dout
BedProperties, 24
_drive_coef
CoupledLDF, 39
_drive_var
CoupledLDF, 39
_driving_value
CoupledLDF, 39
LinearDrivingForce, 134
_eb
BedProperties, 24
_enthalpy_1
MagpieAdsorbateProperties, 141
_enthalpy_2
MagpieAdsorbateProperties, 141

_enthalpy_3
MagpieAdsorbateProperties, 141
_enthalpy_4
MagpieAdsorbateProperties, 141
_enthalpy_5
MagpieAdsorbateProperties, 141
_enthalpy_6
MagpieAdsorbateProperties, 141
_entropy_1
MagpieAdsorbateProperties, 141
_entropy_2
MagpieAdsorbateProperties, 142
_entropy_3
MagpieAdsorbateProperties, 142
_entropy_4
MagpieAdsorbateProperties, 142
_entropy_5
MagpieAdsorbateProperties, 142
_entropy_6
MagpieAdsorbateProperties, 142
_epsilon
DGAnisotropicDiffusion, 44
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 54
DGColumnWallHeatFluxLimitedBC, 60
DGFluxLimitedBC, 66
DGHeatFluxLimitedBC, 72
DGMassFluxLimitedBC, 80
_flow_rate
FlowProperties, 99
_gaining
CoupledLDF, 39
LinearDrivingForce, 134
_gas_conc
AdsorbentProperties, 11
BedProperties, 24
FlowProperties, 99
MagpieAdsorbateProperties, 142
TotalColumnPressure, 189
_gas_conc_old
MagpieAdsorbateProperties, 142
_gas_const
FlowProperties.h, 233
_gas_density
DGColumnHeatAdvection, 46
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 72
FlowProperties, 99
GColumnHeatAdvection, 106
_gas_heat_capacity
DGColumnHeatAdvection, 46
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
FlowProperties, 99
GColumnHeatAdvection, 106
_gas_molecular_wieght
FlowProperties, 99
_gas_viscosity
FlowProperties, 99
_heat_retardation
BedHeatAccumulation, 20
FlowProperties, 99
_hs
AdsorbentProperties, 11
_hw
BedProperties, 24
_index
AdsorbentProperties, 12
BedMassAccumulation, 22
BedProperties, 24
DGColumnMassDispersion, 54
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
FlowProperties, 99
GColumnMassDispersion, 114
MAGPIE_Adsorption, 135
MAGPIE_AdsorptionHeat, 137
MAGPIE_Perturbation, 139
MagpieAdsorbateProperties, 142
TotalColumnPressure, 189
_inner_dia
BedProperties, 24
BedWallHeatTransfer, 27
FlowProperties, 99
WallAmbientHeatTransfer, 192
_input_molefraction
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
_input_pressure
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
_input_temperature
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
_length
BedProperties, 25
_macropore_radius
AdsorbentProperties, 12
_magpie_dat
MAGPIE_Adsorption, 135
MAGPIE_AdsorptionHeat, 137
MAGPIE_Perturbation, 139
MagpieAdsorbateProperties, 142
_max_capacity
MagpieAdsorbateProperties, 142
_molar_volume
MagpieAdsorbateProperties, 143
_molecular_diffusion
DGColumnMassDispersion, 54
DGMassFluxBC, 76
DGMassFluxLimitedBC, 80
FlowProperties, 100
GColumnMassDispersion, 114
_molecular_wieght

FlowProperties, 100
_num_sites
MagpieAdsorbateProperties, 143
_outer_dia
BedProperties, 25
BedWallHeatTransfer, 27
WallAmbientHeatTransfer, 192
_pellet_density
AdsorbentProperties, 12
AdsorptionHeatAccumulation, 14
AdsorptionMassTransfer, 15
FlowProperties, 100
_pellet_diameter
AdsorbentProperties, 12
_pellet_heat_capacity
AdsorbentProperties, 12
FlowProperties, 100
_porosity
AdsorptionHeatAccumulation, 14
AdsorptionMassTransfer, 15
BedProperties, 25
FlowProperties, 100
_retardation
BedMassAccumulation, 22
FlowProperties, 100
_rhos
AdsorbentProperties, 12
_rhow
BedProperties, 25
_sigma
DGAnisotropicDiffusion, 44
DGColumnHeatDispersion, 49
DGColumnMassDispersion, 54
DGColumnWallHeatFluxLimitedBC, 60
DGFluxLimitedBC, 66
DGHeatFluxLimitedBC, 73
DGMassFluxLimitedBC, 80
_solid
AdsorptionMassTransfer, 15
_solid_conc
FlowProperties, 100
MAGPIE_AdsorptionHeat, 137
_solid_heat
AdsorptionHeatAccumulation, 14
_solid_heat_old
AdsorptionHeatAccumulation, 14
_solid_old
AdsorptionMassTransfer, 15
_solid_perturb
FlowProperties, 100
_temperature
AdsorbentProperties, 12
BedProperties, 25
FlowProperties, 100
MagpieAdsorbateProperties, 143
TotalColumnPressure, 189
_total_pressure
FlowProperties, 100
MagpieAdsorbateProperties, 143
_u_input
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 66
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 77
DGMassFluxLimitedBC, 80
_var
CoupledLDF, 39
LinearDrivingForce, 134
_vel
DGColumnHeatAdvection, 46
DGColumnMassAdvection, 51
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 77
DGMassFluxLimitedBC, 81
GColumnHeatAdvection, 106
GColumnMassAdvection, 111
_velocity
DGAdvection, 41
DGColumnHeatAdvection, 46
DGColumnMassAdvection, 51
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 66
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 77
DGMassFluxLimitedBC, 81
FlowProperties, 100
GAdvection, 102
GColumnHeatAdvection, 107
GColumnMassAdvection, 111
_vx
DGAdvection, 41
DGColumnHeatAdvection, 46
DGColumnMassAdvection, 51
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 66
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 77
DGMassFluxLimitedBC, 81
GAdvection, 102
GColumnHeatAdvection, 107
GColumnMassAdvection, 111
_vy
DGAdvection, 41
DGColumnHeatAdvection, 46
DGColumnMassAdvection, 51
DGColumnWallHeatFluxBC, 57

DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 66
DGHeatFluxBC, 69
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 77
DGMassFluxLimitedBC, 81
GAdvection, 102
GColumnHeatAdvection, 107
GColumnMassAdvection, 111
_vz
DGAdvection, 41
DGColumnHeatAdvection, 46
DGColumnMassAdvection, 51
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 60
DGFluxBC, 63
DGFluxLimitedBC, 66
DGHeatFluxBC, 70
DGHeatFluxLimitedBC, 73
DGMassFluxBC, 77
DGMassFluxLimitedBC, 81
GAdvection, 102
GColumnHeatAdvection, 107
GColumnMassAdvection, 111
_wall_density
BedProperties, 25
WallHeatAccumulation, 194
_wall_exterior_transfer_coeff
BedProperties, 25
WallAmbientHeatTransfer, 192
_wall_heat_capacity
BedProperties, 25
WallHeatAccumulation, 194
_wall_temp
DGColumnWallHeatFluxBC, 57
DGColumnWallHeatFluxLimitedBC, 61
_y_IC
ConcentrationIC, 37

A
magpie.h, 258
ARNOLDI_DATA, 16
beta, 17
e1, 17
Hkp1, 17
hp1, 17
iter, 17
k, 17
Output, 17
sum, 17
v, 17
Vk, 17
w, 17
yk, 17
activation_energy
SCOPSOWL_PARAM_DATA, 178
SKUA_PARAM, 184
adjoint
Matrix, 146
Adsorbable
SCOPSOWL_PARAM_DATA, 178
SKUA_PARAM, 184
AdsorbentProperties, 10
_binder_fraction, 11
_binder_porosity, 11
_crystal_radius, 11
_gas_conc, 11
_hs, 11
_index, 12
_macropore_radius, 12
_pellet_density, 12
_pellet_diameter, 12
_pellet_heat_capacity, 12
_rhos, 12
_temperature, 12
AdsorbentProperties, 11
AdsorbentProperties, 11
computeQpProperties, 11
AdsorbentProperties.h, 194
validParams< AdsorbentProperties >, 195
AdsorptionHeatAccumulation, 12
_pellet_density, 14
_porosity, 14
_solid_heat, 14
_solid_heat_old, 14
AdsorptionHeatAccumulation, 13
AdsorptionHeatAccumulation, 13
computeQpJacobian, 13
computeQpResidual, 13
AdsorptionHeatAccumulation.h, 195
validParams< AdsorptionHeatAccumulation >, 196
AdsorptionMassTransfer, 14
_pellet_density, 15
_porosity, 15
_solid, 15
_solid_old, 15
AdsorptionMassTransfer, 15
AdsorptionMassTransfer, 15
computeQpJacobian, 15
computeQpResidual, 15
AdsorptionMassTransfer.h, 196
validParams< AdsorptionMassTransfer >, 197
affinity
SCOPSOWL_PARAM_DATA, 178
SKUA_PARAM, 184
Ai
OPTRANS_DATA, 159
alpha
BACKTRACK_DATA, 18
BiCGSTAB_DATA, 29
CGS_DATA, 32
GCR_DATA, 115
PCG_DATA, 160
anchor_alias_dne
error.h, 225

Ap
PCG_DATA, 160
arg
GMRESR_DATA, 121
arg_matrix_same
error.h, 224
arnoldi
lark.h, 243
arnoldi_dat
GMRESLP_DATA, 119
As
SYSTEM_DATA, 186
associateSyntax
DgospreyApp, 82
avg_norm
SYSTEM_DATA, 186
avgDp
scopsowl.h, 269

BACKTRACK_DATA, 18
alpha, 18
constRho, 18
Fk, 19
fun_call, 19
lambdaMin, 19
normFkp1, 19
rho, 19
xk, 19
backtrack_dat
PJFNK_DATA, 166
backtrackLineSearch
lark.h, 243
base/DgospreyRevision.h
DGOSPREY_REVISION, 218
BedHeatAccumulation, 19
_heat_retardation, 20
BedHeatAccumulation, 20
BedHeatAccumulation, 20
computeQpJacobian, 20
computeQpResidual, 20
BedHeatAccumulation.h, 197
validParams< BedHeatAccumulation >, 198
BedMassAccumulation, 21
_index, 22
_retardation, 22
BedMassAccumulation, 21
BedMassAccumulation, 21
computeQpJacobian, 22
computeQpResidual, 22
BedMassAccumulation.h, 198
validParams< BedMassAccumulation >, 199
BedProperties, 22
_Kz, 25
_Ua, 25
_Uw, 25
_bed_wall_transfer_coeff, 24
_conductivity, 24
_din, 24
_dout, 24
_eb, 24
_gas_conc, 24
_hw, 24
_index, 24
_inner_dia, 24
_length, 25
_outer_dia, 25
_porosity, 25
_rhow, 25
_temperature, 25
_wall_density, 25
_wall_exterior_transfer_coeff, 25
_wall_heat_capacity, 25
BedProperties, 24
BedProperties, 24
computeQpProperties, 24
BedProperties.h, 199
validParams< BedProperties >, 199
BedWallHeatTransfer, 26
_bed_wall_transfer_coeff, 27
_column_temp, 27
_inner_dia, 27
_outer_dia, 27
BedWallHeatTransfer, 27
BedWallHeatTransfer, 27
computeQpJacobian, 27
computeQpResidual, 27
BedWallHeatTransfer.h, 199
validParams< BedWallHeatTransfer >, 200
bestres
BiCGSTAB_DATA, 29
CGS_DATA, 32
GCR_DATA, 115
GMRESLP_DATA, 119
GMRESRP_DATA, 124
PCG_DATA, 160
PICARD_DATA, 163
bestx
BiCGSTAB_DATA, 29
CGS_DATA, 33
GCR_DATA, 116
GMRESLP_DATA, 119
GMRESRP_DATA, 124
PCG_DATA, 161
PICARD_DATA, 163
PJFNK_DATA, 166
beta
ARNOLDI_DATA, 17
BiCGSTAB_DATA, 29
CGS_DATA, 33
FINCH_DATA, 87
GCR_DATA, 116
PCG_DATA, 161
BiCGSTAB
lark.h, 243
BiCGSTAB_DATA, 27
alpha, 29
bestres, 29

bestx, 29
beta, 29
breakdown, 29
iter, 29
maxit, 29
omega, 29
omega_old, 29
Output, 30
p, 30
r, 30
r0, 30
relres, 30
relres_base, 30
res, 30
rho, 30
rho_old, 30
s, 30
t, 30
tol_abs, 30
tol_rel, 31
v, 31
x, 31
y, 31
z, 31
bicgstab
lark.h, 244
bicgstab_dat
PJFNK_DATA, 166
binary_diffusion
MIXED_GAS, 155
binder_fraction
SCOPSOWL_DATA, 173
binder_poresize
SCOPSOWL_DATA, 173
binder_porosity
SCOPSOWL_DATA, 173
Bounce
PJFNK_DATA, 166
breakdown
BiCGSTAB_DATA, 29
CGS_DATA, 33
GCR_DATA, 116

c
CGS_DATA, 33
GCR_DATA, 116
CGS
lark.h, 243
c_temp
GCR_DATA, 116
CC_E
FINCH_DATA, 87
CC_I
FINCH_DATA, 87
CE3
egret.h, 220
CGS_DATA, 31
alpha, 32
bestres, 32
bestx, 33
beta, 33
breakdown, 33
c, 33
iter, 33
maxit, 33
Output, 33
p, 33
r, 33
r0, 33
relres, 33
relres_base, 34
res, 34
rho, 34
sigma, 34
tol_abs, 34
tol_rel, 34
u, 34
v, 34
w, 34
x, 34
z, 34
CL_E
FINCH_DATA, 87
CL_I
FINCH_DATA, 87
CN
FINCH_DATA, 87
COUPLEDLDF_H
CoupledLDF.h, 203
CR_E
FINCH_DATA, 88
CR_I
FINCH_DATA, 88
calculate_properties
egret.h, 222
callroutine
FINCH_DATA, 87
Carrier
SYSTEM_DATA, 186
Cartesian
finch.h, 228
cgs
lark.h, 245
cgs_dat
PJFNK_DATA, 166
char_length
MIXED_GAS, 155
char_macro
SCOPSOWL_DATA, 173
char_measure
SKUA_DATA, 181
char_micro
SCOPSOWL_DATA, 173
check_Mass
finch.h, 228
CheckMass
FINCH_DATA, 87

CheckMolefractions
MIXED_GAS, 155
cofactor
Matrix, 146
columnExtend
Matrix, 146
columnExtract
Matrix, 147
columnProjection
Matrix, 147
columnReplace
Matrix, 147
columnShrink
Matrix, 147
ColumnTemperatureIC, 35
_TC_IC, 36
ColumnTemperatureIC, 35
ColumnTemperatureIC, 35
value, 36
ColumnTemperatureIC.h, 200
validParams< ColumnTemperatureIC >, 201
columnVectorFill
Matrix, 147
columns
Matrix, 147
computeQpJacobian
AdsorptionHeatAccumulation, 13
AdsorptionMassTransfer, 15
BedHeatAccumulation, 20
BedMassAccumulation, 22
BedWallHeatTransfer, 27
CoupledLDF, 39
DGAdvection, 41
DGAnisotropicDiffusion, 43
DGColumnHeatAdvection, 45
DGColumnHeatDispersion, 48
DGColumnMassAdvection, 50
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
DGFluxBC, 62
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 71
DGMassFluxBC, 75
DGMassFluxLimitedBC, 79
GAdvection, 102
GAnisotropicDiffusion, 104
GColumnHeatAdvection, 106
GColumnHeatDispersion, 108
GColumnMassAdvection, 111
GColumnMassDispersion, 113
LinearDrivingForce, 133
WallAmbientHeatTransfer, 192
WallHeatAccumulation, 193
computeQpProperties
AdsorbentProperties, 11
BedProperties, 24
FlowProperties, 98
MagpieAdsorbateProperties, 141
computeQpResidual
AdsorptionHeatAccumulation, 13
AdsorptionMassTransfer, 15
BedHeatAccumulation, 20
BedMassAccumulation, 22
BedWallHeatTransfer, 27
CoupledLDF, 39
DGAdvection, 41
DGAnisotropicDiffusion, 43
DGColumnHeatAdvection, 45
DGColumnHeatDispersion, 48
DGColumnMassAdvection, 50
DGColumnMassDispersion, 53
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxLimitedBC, 59
DGFluxBC, 62
DGFluxLimitedBC, 65
DGHeatFluxBC, 68
DGHeatFluxLimitedBC, 71
DGMassFluxBC, 75
DGMassFluxLimitedBC, 79
GAdvection, 102
GAnisotropicDiffusion, 104
GColumnHeatAdvection, 106
GColumnHeatDispersion, 108
GColumnMassAdvection, 111
GColumnMassDispersion, 113
LinearDrivingForce, 133
WallAmbientHeatTransfer, 192
WallHeatAccumulation, 193
computeValue
MAGPIE_Adsorption, 135
MAGPIE_AdsorptionHeat, 136
MAGPIE_Perturbation, 139
TotalColumnPressure, 189
ConcentrationIC, 36
_PT_IC, 37
_T_IC, 37
_y_IC, 37
ConcentrationIC, 37
ConcentrationIC, 37
value, 37
ConcentrationIC.h, 201
validParams< ConcentrationIC >, 202
const_Dc
skua.h, 276
const_filmMassTransfer
scopsowl.h, 269
const_kf
skua.h, 276
const_pore_diffusion
scopsowl.h, 269
constRho
BACKTRACK_DATA, 18
ConstantICFill
Matrix, 147

coord
SKUA_DATA, 181
coord_macro
SCOPSOWL_DATA, 173
coord_micro
SCOPSOWL_DATA, 173
CoupledLDF, 37
_coef, 39
_drive_coef, 39
_drive_var, 39
_driving_value, 39
_gaining, 39
_var, 39
computeQpJacobian, 39
computeQpResidual, 39
CoupledLDF, 39
CoupledLDF, 39
CoupledLDF.h, 202
COUPLEDLDF_H, 203
validParams< CoupledLDF >, 203
crystal_radius
SCOPSOWL_DATA, 173
Cstd
egret.h, 220
Cylindrical
finch.h, 228

d
FINCH_DATA, 88
D_c
skua.h, 275
D_ii
egret.h, 220
D_ij
egret.h, 220
D_inf
skua.h, 275
D_o
skua.h, 275
DBL_EPSILON
magpie.h, 258
DGAdvection, 40
_velocity, 41
_vx, 41
_vy, 41
_vz, 41
computeQpJacobian, 41
computeQpResidual, 41
DGAdvection, 41
DGAdvection, 41
DGAdvection.h, 203
validParams< DGAdvection >, 204
DGAnisotropicDiffusion, 42
_Diffusion, 43
_Dxx, 43
_Dxy, 43
_Dxz, 43
_Dyx, 43
_Dyy, 43
_Dyz, 43
_Dzx, 44
_Dzy, 44
_Dzz, 44
_epsilon, 44
_sigma, 44
computeQpJacobian, 43
computeQpResidual, 43
DGAnisotropicDiffusion, 43
DGAnisotropicDiffusion, 43
DGAnisotropicDiffusion.h, 204
validParams< DGAnisotropicDiffusion >, 205
DGColumnHeatAdvection, 44
_gas_density, 46
_gas_heat_capacity, 46
_vel, 46
_velocity, 46
_vx, 46
_vy, 46
_vz, 46
computeQpJacobian, 45
computeQpResidual, 45
DGColumnHeatAdvection, 45
DGColumnHeatAdvection, 45
DGColumnHeatAdvection.h, 205
validParams< DGColumnHeatAdvection >, 206
DGColumnHeatDispersion, 46
_Diffusion, 48
_Dxx, 48
_Dxy, 48
_Dxz, 48
_Dyx, 48
_Dyy, 49
_Dyz, 49
_Dzx, 49
_Dzy, 49
_Dzz, 49
_conductivity, 48
_epsilon, 49
_sigma, 49
computeQpJacobian, 48
computeQpResidual, 48
DGColumnHeatDispersion, 48
DGColumnHeatDispersion, 48
DGColumnHeatDispersion.h, 206
validParams< DGColumnHeatDispersion >, 207
DGColumnMassAdvection, 49
_vel, 51
_velocity, 51
_vx, 51
_vy, 51
_vz, 51
computeQpJacobian, 50
computeQpResidual, 50
DGColumnMassAdvection, 50
DGColumnMassAdvection, 50
DGColumnMassAdvection.h, 208
validParams< DGColumnMassAdvection >, 208

DGColumnMassDispersion, 51
_Diffusion, 53
_Dxx, 53
_Dxy, 53
_Dxz, 53
_Dyx, 53
_Dyy, 53
_Dyz, 53
_Dzx, 54
_Dzy, 54
_Dzz, 54
_dispersion, 53
_epsilon, 54
_index, 54
_molecular_diffusion, 54
_sigma, 54
computeQpJacobian, 53
computeQpResidual, 53
DGColumnMassDispersion, 53
DGColumnMassDispersion, 53
DGColumnMassDispersion.h, 209
validParams< DGColumnMassDispersion >, 210
DGColumnWallHeatFluxBC, 54
_Diffusion, 56
_Dxx, 56
_Dxy, 56
_Dxz, 56
_Dyx, 56
_Dyy, 56
_Dyz, 57
_Dzx, 57
_Dzy, 57
_Dzz, 57
_bed_wall_transfer_coeff, 56
_conductivity, 56
_u_input, 57
_velocity, 57
_vx, 57
_vy, 57
_vz, 57
_wall_temp, 57
computeQpJacobian, 56
computeQpResidual, 56
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxBC, 56
DGColumnWallHeatFluxBC.h, 210
validParams< DGColumnWallHeatFluxBC >, 210
DGColumnWallHeatFluxLimitedBC, 57
_Diffusion, 59
_Dxx, 59
_Dxy, 59
_Dxz, 60
_Dyx, 60
_Dyy, 60
_Dyz, 60
_Dzx, 60
_Dzy, 60
_Dzz, 60
_bed_wall_transfer_coeff, 59
_conductivity, 59
_epsilon, 60
_sigma, 60
_u_input, 60
_velocity, 60
_vx, 60
_vy, 60
_vz, 60
_wall_temp, 61
computeQpJacobian, 59
computeQpResidual, 59
DGColumnWallHeatFluxLimitedBC, 59
DGColumnWallHeatFluxLimitedBC, 59
DGColumnWallHeatFluxLimitedBC.h, 211
validParams< DGColumnWallHeatFluxLimitedBC >, 211
DGFluxBC, 61
_Diffusion, 62
_Dxx, 62
_Dxy, 62
_Dxz, 62
_Dyx, 63
_Dyy, 63
_Dyz, 63
_Dzx, 63
_Dzy, 63
_Dzz, 63
_u_input, 63
_velocity, 63
_vx, 63
_vy, 63
_vz, 63
computeQpJacobian, 62
computeQpResidual, 62
DGFluxBC, 62
DGFluxBC, 62
DGFluxBC.h, 211
validParams< DGFluxBC >, 212
DGFluxLimitedBC, 63
_Diffusion, 65
_Dxx, 65
_Dxy, 65
_Dxz, 65
_Dyx, 65
_Dyy, 65
_Dyz, 65
_Dzx, 65
_Dzy, 65
_Dzz, 66
_epsilon, 66
_sigma, 66
_u_input, 66
_velocity, 66
_vx, 66
_vy, 66
_vz, 66
computeQpJacobian, 65

computeQpResidual, 65
DGFluxLimitedBC, 65
DGFluxLimitedBC, 65
DGFluxLimitedBC.h, 212
validParams< DGFluxLimitedBC >, 213
DGHeatFluxBC, 66
_Diffusion, 68
_Dxx, 68
_Dxy, 68
_Dxz, 68
_Dyx, 68
_Dyy, 69
_Dyz, 69
_Dzx, 69
_Dzy, 69
_Dzz, 69
_conductivity, 68
_gas_density, 69
_gas_heat_capacity, 69
_input_temperature, 69
_u_input, 69
_vel, 69
_velocity, 69
_vx, 69
_vy, 69
_vz, 70
computeQpJacobian, 68
computeQpResidual, 68
DGHeatFluxBC, 68
DGHeatFluxBC, 68
DGHeatFluxBC.h, 213
validParams< DGHeatFluxBC >, 214
DGHeatFluxLimitedBC, 70
_Diffusion, 72
_Dxx, 72
_Dxy, 72
_Dxz, 72
_Dyx, 72
_Dyy, 72
_Dyz, 72
_Dzx, 72
_Dzy, 72
_Dzz, 72
_conductivity, 72
_epsilon, 72
_gas_density, 72
_gas_heat_capacity, 73
_input_temperature, 73
_sigma, 73
_u_input, 73
_vel, 73
_velocity, 73
_vx, 73
_vy, 73
_vz, 73
computeQpJacobian, 71
computeQpResidual, 71
DGHeatFluxLimitedBC, 71
DGHeatFluxLimitedBC, 71
DGHeatFluxLimitedBC.h, 214
validParams< DGHeatFluxLimitedBC >, 215
DGMassFluxBC, 73
_Diffusion, 75
_Dxx, 75
_Dxy, 76
_Dxz, 76
_Dyx, 76
_Dyy, 76
_Dyz, 76
_Dzx, 76
_Dzy, 76
_Dzz, 76
_dispersion, 75
_index, 76
_input_molefraction, 76
_input_pressure, 76
_input_temperature, 76
_molecular_diffusion, 76
_u_input, 77
_vel, 77
_velocity, 77
_vx, 77
_vy, 77
_vz, 77
computeQpJacobian, 75
computeQpResidual, 75
DGMassFluxBC, 75
DGMassFluxBC, 75
DGMassFluxBC.h, 215
validParams< DGMassFluxBC >, 216
DGMassFluxLimitedBC, 77
_Diffusion, 79
_Dxx, 79
_Dxy, 79
_Dxz, 79
_Dyx, 79
_Dyy, 80
_Dyz, 80
_Dzx, 80
_Dzy, 80
_Dzz, 80
_dispersion, 79
_epsilon, 80
_index, 80
_input_molefraction, 80
_input_pressure, 80
_input_temperature, 80
_molecular_diffusion, 80
_sigma, 80
_u_input, 80
_vel, 81
_velocity, 81
_vx, 81
_vy, 81
_vz, 81
computeQpJacobian, 79

computeQpResidual, 79
DGMassFluxLimitedBC, 79
DGMassFluxLimitedBC, 79
DGMassFluxLimitedBC.h, 216
validParams< DGMassFluxLimitedBC >, 217
DGOSPREY_REVISION
base/DgospreyRevision.h, 218
DgospreyRevision.h, 218
dHo
GSTA_DATA, 129
DIC
FINCH_DATA, 88
dSo
GSTA_DATA, 129
Data
Matrix, 153
default_Dc
skua.h, 276
default_adsorption
scopsowl.h, 270
default_bcs
finch.h, 228
default_effective_diffusion
scopsowl.h, 270
default_execution
finch.h, 229
default_filmMassTransfer
scopsowl.h, 270
default_ic
finch.h, 229
default_kf
skua.h, 276
default_params
finch.h, 229
default_pore_diffusion
scopsowl.h, 270
default_postprocess
finch.h, 229
default_precon
finch.h, 229
default_preprocess
finch.h, 229
default_res
finch.h, 229
default_reset
finch.h, 229
default_retardation
scopsowl.h, 270
default_solve
finch.h, 229
default_surf_diffusion
scopsowl.h, 271
default_timestep
finch.h, 230
density
PURE_GAS, 170
determinate
Matrix, 147
DgospreyApp, 81
~DgospreyApp, 82
associateSyntax, 82
DgospreyApp, 82
DgospreyApp, 82
registerApps, 82
registerObjects, 82
DgospreyApp.h, 217
validParams< DgospreyApp >, 218
DgospreyRevision.h, 218
DGOSPREY_REVISION, 218
diagonalSolve
Matrix, 148
dim_mis_match
error.h, 224
Dirichlet
FINCH_DATA, 88
DirichletBC
SCOPSOWL_DATA, 173
SKUA_DATA, 181
dirichletBCFill
Matrix, 148
discretize
FINCH_DATA, 88
Display
Matrix, 148
Dk
scopsowl.h, 269
Dn
FINCH_DATA, 88
Dnp1
FINCH_DATA, 88
Do
FINCH_DATA, 88
Dp
scopsowl.h, 269
Dp_ij
egret.h, 220
dq_dc
SCOPSOWL_PARAM_DATA, 178
dq_dco
SCOPSOWL_PARAM_DATA, 178
dq_dp
magpie.h, 259
dt
FINCH_DATA, 88
dt_old
FINCH_DATA, 88
duplicate_variable
error.h, 225
dxj
NUM_JAC_DATA, 158
dynamic_viscosity
PURE_GAS, 170
dz
FINCH_DATA, 89

e0
GMRESRP_DATA, 125

e0_bar
GMRESRP_DATA, 125
e1
ARNOLDI_DATA, 17
EGRET_HPP_
egret.h, 220
eMax
magpie.h, 259
mSPD_DATA, 157
edit
Matrix, 148
egret.h, 218
CE3, 220
calculate_properties, 222
Cstd, 220
D_ii, 220
D_ij, 220
Dp_ij, 220
EGRET_HPP_, 220
FilmMTCoeff, 220
initialize_data, 222
Mu, 221
Nu, 221
PE3, 221
PSI, 221
Po, 221
Pstd, 221
RE3, 221
ReNum, 221
Rstd, 221
ScNum, 221
set_variables, 222
empirical_kf
skua.h, 277
empty_matrix
error.h, 224
eps
NUM_JAC_DATA, 158
PJFNK_DATA, 166
error
error.h, 225
error.h
anchor_alias_dne, 225
arg_matrix_same, 224
dim_mis_match, 224
duplicate_variable, 225
empty_matrix, 224
file_dne, 224
generic_error, 224
indexing_error, 224
initial_error, 225
invalid_atom, 225
invalid_boolean, 224
invalid_components, 224
invalid_console_input, 225
invalid_electron, 225
invalid_fraction, 224
invalid_gas_sum, 224
invalid_molefraction, 224
invalid_neutron, 225
invalid_norm, 225
invalid_proton, 225
invalid_size, 224
invalid_solid_sum, 224
invalid_species, 225
invalid_type, 225
invalid_valence, 225
key_not_found, 225
magpie_reverse_error, 224
matrix_too_small, 224
matvec_mis_match, 224
missing_information, 225
negative_mass, 224
negative_time, 224
no_diffusion, 224
non_real_edge, 225
non_square_matrix, 224
not_a_token, 225
nullptr_error, 225
nullptr_func, 224
opt_no_support, 224
ortho_check_fail, 224
out_of_bounds, 224
read_error, 225
rxn_rate_error, 225
scenario_fail, 224
simulation_fail, 224
singular_matrix, 224
string_parse_error, 225
tensor_out_of_bounds, 225
unregistered_name, 225
unstable_matrix, 224
vector_out_of_bounds, 225
zero_vector, 225
error.h, 222
error, 225
error_type, 224
mError, 223
error_type
error.h, 224
eta
mSPD_DATA, 157
eval_GPAST
magpie.h, 259
eval_ads
SCOPSOWL_DATA, 174
eval_diff
SCOPSOWL_DATA, 174
SKUA_DATA, 181
eval_eta
magpie.h, 259
eval_kf
SCOPSOWL_DATA, 174
SKUA_DATA, 181
eval_po
magpie.h, 260

eval_po_PI
magpie.h, 260
eval_po_qo
magpie.h, 260
eval_retard
SCOPSOWL_DATA, 174
eval_surfDiff
SCOPSOWL_DATA, 174
evalprecon
FINCH_DATA, 89
evalres
FINCH_DATA, 89
ExplicitFlux
FINCH_DATA, 89

F
PJFNK_DATA, 166
FINCH_Picard
finch.h, 228
FOM
lark.h, 243
fC_E
FINCH_DATA, 89
fC_I
FINCH_DATA, 89
FINCH_DATA, 82
beta, 87
CC_E, 87
CC_I, 87
CL_E, 87
CL_I, 87
CN, 87
CR_E, 88
CR_I, 88
callroutine, 87
CheckMass, 87
d, 88
DIC, 88
Dirichlet, 88
discretize, 88
Dn, 88
Dnp1, 88
Do, 88
dt, 88
dt_old, 88
dz, 89
evalprecon, 89
evalres, 89
ExplicitFlux, 89
fC_E, 89
fC_I, 89
fL_E, 89
fL_I, 89
fR_E, 89
fR_I, 90
Fn, 89
Fnp1, 89
gE, 90
gI, 90
Iterative, 90
kIC, 90
kfn, 90
kfnp1, 90
kn, 90
knp1, 90
ko, 90
L, 90
LN, 91
lambda_E, 91
lambda_I, 91
ME, 91
MI, 91
max_iter, 91
NE, 91
NI, 91
nl_method, 91
NormTrack, 91
OE, 91
OI, 92
param_data, 92
picard_dat, 92
pjfnk_dat, 92
pres, 92
RIC, 92
res, 92
resettime, 92
Rn, 92
Rnp1, 92
Ro, 92
s, 93
setbcs, 93
setic, 93
setparams, 93
setpostprocess, 93
setpreprocess, 93
settime, 93
Sn, 93
Snp1, 93
solve, 93
SteadyState, 93
T, 94
t, 94
t_old, 94
tol_abs, 94
tol_rel, 94
total_iter, 94
u_star, 94
uAvg, 94
uAvg_old, 94
uIC, 94
uT, 95
uT_old, 95
ubest, 94
un, 95
unm1, 95
unp1, 95
uo, 95

Update, 95
uz_l_E, 95
uz_l_I, 95
uz_lm1_E, 95
uz_lm1_I, 95
uz_lp1_E, 95
uz_lp1_I, 96
vIC, 96
vn, 96
vnp1, 96
vo, 96
fL_E
FINCH_DATA, 89
fL_I
FINCH_DATA, 89
fR_E
FINCH_DATA, 89
fR_I
FINCH_DATA, 90
file_dne
error.h, 224
film_transfer
SCOPSOWL_PARAM_DATA, 178
SKUA_PARAM, 184
FilmMTCoeff
egret.h, 220
finch.h
Cartesian, 228
Cylindrical, 228
FINCH_Picard, 228
LARK_PJFNK, 228
LARK_Picard, 228
Spherical, 228
finch.h, 225
check_Mass, 228
default_bcs, 228
default_execution, 229
default_ic, 229
default_params, 229
default_postprocess, 229
default_precon, 229
default_preprocess, 229
default_res, 229
default_reset, 229
default_solve, 229
default_timestep, 230
finch_coord_type, 228
finch_solve_type, 228
l_direct, 230
lark_picard_step, 230
max, 230
min, 230
minmod, 230
minmod_discretization, 230
nl_picard, 230
ospre_discretization, 230
print2file_dim_header, 230
print2file_newline, 230
print2file_result_new, 231
print2file_result_old, 231
print2file_tab, 231
print2file_time_header, 231
setup_FINCH_DATA, 231
uAverage, 231
uTotal, 231
vanAlbada_discretization, 232
finch_coord_type
finch.h, 228
finch_dat
SCOPSOWL_DATA, 174
SKUA_DATA, 182
finch_solve_type
finch.h, 228
Fk
BACKTRACK_DATA, 19
flock.h, 232
FlowProperties, 96
_column_length, 98
_comp_Sutherland_const, 99
_comp_heat_capacity, 98
_comp_ref_temp, 98
_comp_ref_viscosity, 98
_dispersion, 99
_flow_rate, 99
_gas_conc, 99
_gas_density, 99
_gas_heat_capacity, 99
_gas_molecular_wieght, 99
_gas_viscosity, 99
_heat_retardation, 99
_index, 99
_inner_dia, 99
_molecular_diffusion, 100
_molecular_wieght, 100
_pellet_density, 100
_pellet_heat_capacity, 100
_porosity, 100
_retardation, 100
_solid_conc, 100
_solid_perturb, 100
_temperature, 100
_total_pressure, 100
_velocity, 100
computeQpProperties, 98
FlowProperties, 98
FlowProperties, 98
FlowProperties.h, 232
_gas_const, 233
validParams< FlowProperties >, 234
Fn
FINCH_DATA, 89
Fnp1
FINCH_DATA, 89
fom
lark.h, 245
fun_call

BACKTRACK_DATA, 19
PJFNK_DATA, 166
funeval
PJFNK_DATA, 167
Fv
PJFNK_DATA, 167
Fx
NUM_JAC_DATA, 158
Fxp
NUM_JAC_DATA, 158

GCR
lark.h, 243
GMRESLP
lark.h, 243
GMRESR
lark.h, 243
GMRESRP
lark.h, 243
GAdvection, 101
_velocity, 102
_vx, 102
_vy, 102
_vz, 102
computeQpJacobian, 102
computeQpResidual, 102
GAdvection, 102
GAdvection, 102
GAdvection.h, 234
validParams< GAdvection >, 234
GAnisotropicDiffusion, 103
_Diffusion, 104
_Dxx, 104
_Dxy, 104
_Dxz, 104
_Dyx, 104
_Dyy, 104
_Dyz, 104
_Dzx, 104
_Dzy, 104
_Dzz, 105
computeQpJacobian, 104
computeQpResidual, 104
GAnisotropicDiffusion, 104
GAnisotropicDiffusion, 104
GAnisotropicDiffusion.h, 234
validParams< GAnisotropicDiffusion >, 235
GCR_DATA, 114
alpha, 115
bestres, 115
bestx, 116
beta, 116
breakdown, 116
c, 116
c_temp, 116
iter_inner, 116
iter_outer, 116
maxit, 116
Output, 116
r, 116
relres, 116
relres_base, 117
res, 117
restart, 117
tol_abs, 117
tol_rel, 117
total_iter, 117
transpose_dat, 117
u, 117
u_temp, 117
x, 117
GCR_Output
GMRESR_DATA, 121
GColumnHeatAdvection, 105
_gas_density, 106
_gas_heat_capacity, 106
_vel, 106
_velocity, 107
_vx, 107
_vy, 107
_vz, 107
computeQpJacobian, 106
computeQpResidual, 106
GColumnHeatAdvection, 106
GColumnHeatAdvection, 106
GColumnHeatAdvection.h, 235
validParams< GColumnHeatAdvection >, 236
GColumnHeatDispersion, 107
_Diffusion, 109
_Dxx, 109
_Dxy, 109
_Dxz, 109
_Dyx, 109
_Dyy, 109
_Dyz, 109
_Dzx, 109
_Dzy, 109
_Dzz, 109
_conductivity, 109
computeQpJacobian, 108
computeQpResidual, 108
GColumnHeatDispersion, 108
GColumnHeatDispersion, 108
GColumnHeatDispersion.h, 236
validParams< GColumnHeatDispersion >, 237
GColumnMassAdvection, 109
_vel, 111
_velocity, 111
_vx, 111
_vy, 111
_vz, 111
computeQpJacobian, 111
computeQpResidual, 111
GColumnMassAdvection, 110
GColumnMassAdvection, 110
GColumnMassAdvection.h, 237
validParams< GColumnMassAdvection >, 238

GColumnMassDispersion, 111
_Diffusion, 113
_Dxx, 113
_Dxy, 113
_Dxz, 113
_Dyx, 113
_Dyy, 114
_Dyz, 114
_Dzx, 114
_Dzy, 114
_Dzz, 114
_dispersion, 113
_index, 114
_molecular_diffusion, 114
computeQpJacobian, 113
computeQpResidual, 113
GColumnMassDispersion, 113
GColumnMassDispersion, 113
GColumnMassDispersion.h, 238
validParams< GColumnMassDispersion >, 239
gE
FINCH_DATA, 90
gI
FINCH_DATA, 90
GMRES_Output
GMRESR_DATA, 122
GMRESLP_DATA, 118
arnoldi_dat, 119
bestres, 119
bestx, 119
iter, 119
maxit, 119
Output, 119
r, 119
relres, 119
relres_base, 119
res, 119
restart, 119
steps, 119
tol_abs, 120
tol_rel, 120
x, 120
GMRESR_DATA, 120
arg, 121
GCR_Output, 121
GMRES_Output, 122
gcr_abs_tol, 121
gcr_dat, 121
gcr_maxit, 121
gcr_rel_tol, 122
gcr_restart, 122
gmres_dat, 122
gmres_maxit, 122
gmres_restart, 122
gmres_tol, 122
iter_inner, 122
iter_outer, 122
matvec, 122
matvec_data, 122
N, 123
term_precon, 123
terminal_precon, 123
total_iter, 123
GMRESRP_DATA, 123
bestres, 124
bestx, 124
e0, 125
e0_bar, 125
H, 125
H_bar, 125
iter_inner, 125
iter_outer, 125
iter_total, 125
maxit, 125
Output, 125
r, 125
relres, 125
relres_base, 126
res, 126
restart, 126
sum, 126
tol_abs, 126
tol_rel, 126
v, 126
Vk, 126
w, 126
x, 126
y, 126
Zk, 127
GPAST_DATA, 127
gama_inf, 128
He, 128
Plo, 128
po, 128
poi, 128
present, 128
q, 128
qo, 128
x, 128
y, 128
GSTA_DATA, 129
dHo, 129
dSo, 129
m, 129
qmax, 129
gama
mSPD_DATA, 157
gama_inf
GPAST_DATA, 128
gas_dat
SCOPSOWL_DATA, 174
SKUA_DATA, 182
gas_temperature
MIXED_GAS, 155
SCOPSOWL_DATA, 174
gas_velocity

SCOPSOWL_DATA, 174
SKUA_DATA, 182
gcr
lark.h, 246
gcr_abs_tol
GMRESR_DATA, 121
gcr_dat
GMRESR_DATA, 121
PJFNK_DATA, 167
gcr_maxit
GMRESR_DATA, 121
gcr_rel_tol
GMRESR_DATA, 122
gcr_restart
GMRESR_DATA, 122
generic_error
error.h, 224
gmres_dat
GMRESR_DATA, 122
gmres_in
KMS_DATA, 131
gmres_maxit
GMRESR_DATA, 122
gmres_out
KMS_DATA, 131
gmres_restart
GMRESR_DATA, 122
gmres_tol
GMRESR_DATA, 122
gmresLeftPreconditioned
lark.h, 247
gmresRightPreconditioned
lark.h, 248
gmreslp_dat
PJFNK_DATA, 167
gmresr
lark.h, 247
gmresr_dat
PJFNK_DATA, 167
gmresrPreconditioner
lark.h, 249
gmresrp_dat
PJFNK_DATA, 167
gpast_dat
MAGPIE_DATA, 137
grad_mSPD
magpie.h, 260
gsta_dat
MAGPIE_DATA, 137

H
GMRESRP_DATA, 125
H_bar
GMRESRP_DATA, 125
He
GPAST_DATA, 128
magpie.h, 258
Heterogeneous
SCOPSOWL_DATA, 174
Hkp1
ARNOLDI_DATA, 17
hp1
ARNOLDI_DATA, 17

I
SYSTEM_DATA, 186
Ideal
SYSTEM_DATA, 186
Ii
OPTRANS_DATA, 159
indexing_error
error.h, 224
initial_error
error.h, 225
initialGuess_mSPD
magpie.h, 261
initialize_data
egret.h, 222
inner_iter
KMS_DATA, 131
inner_product
Matrix, 148
inner_reltol
KMS_DATA, 131
IntegralAvg
Matrix, 148
IntegralTotal
Matrix, 149
invalid_atom
error.h, 225
invalid_boolean
error.h, 224
invalid_components
error.h, 224
invalid_console_input
error.h, 225
invalid_electron
error.h, 225
invalid_fraction
error.h, 224
invalid_gas_sum
error.h, 224
invalid_molefraction
error.h, 224
invalid_neutron
error.h, 225
invalid_norm
error.h, 225
invalid_proton
error.h, 225
invalid_size
error.h, 224
invalid_solid_sum
error.h, 224
invalid_species
error.h, 225
invalid_type
error.h, 225

invalid_valence
error.h, 225
inverse
Matrix, 149
iter
ARNOLDI_DATA, 17
BiCGSTAB_DATA, 29
CGS_DATA, 33
GMRESLP_DATA, 119
PCG_DATA, 161
PICARD_DATA, 163
iter_inner
GCR_DATA, 116
GMRESR_DATA, 122
GMRESRP_DATA, 125
iter_outer
GCR_DATA, 116
GMRESR_DATA, 122
GMRESRP_DATA, 125
iter_total
GMRESRP_DATA, 125
Iterative
FINCH_DATA, 90

J
SYSTEM_DATA, 187
jacvec
lark.h, 249

K
SYSTEM_DATA, 187
k
ARNOLDI_DATA, 17
kB
magpie.h, 258
kIC
FINCH_DATA, 90
KMS_DATA, 129
gmres_in, 131
gmres_out, 131
inner_iter, 131
inner_reltol, 131
level, 131
matvec, 131
matvec_data, 131
max_level, 131
maxit, 131
outer_abstol, 131
outer_iter, 131
outer_reltol, 131
Output_in, 132
Output_out, 132
restart, 132
term_precon, 132
terminal_precon, 132
total_iter, 132
key_not_found
error.h, 225
kfn
FINCH_DATA, 90
kfnp1
FINCH_DATA, 90
kinematic_viscosity
MIXED_GAS, 155
kmsPreconditioner
lark.h, 249
kn
FINCH_DATA, 90
knp1
FINCH_DATA, 90
ko
FINCH_DATA, 90
krylov_method
lark.h, 242
krylovMultiSpace
lark.h, 249

L
FINCH_DATA, 90
LARK_PJFNK
finch.h, 228
LARK_Picard
finch.h, 228
L_Output
PJFNK_DATA, 167
l_direct
finch.h, 230
l_iter
PJFNK_DATA, 167
LN
FINCH_DATA, 91
ladshawSolve
Matrix, 149
lambda_E
FINCH_DATA, 91
lambda_I
FINCH_DATA, 91
lambdaMin
BACKTRACK_DATA, 19
lark.h
BiCGSTAB, 243
CGS, 243
FOM, 243
GCR, 243
GMRESLP, 243
GMRESR, 243
GMRESRP, 243
PCG, 243
lark.h, 239
arnoldi, 243
backtrackLineSearch, 243
bicgstab, 244
cgs, 245
fom, 245
gcr, 246
gmresLeftPreconditioned, 247
gmresRightPreconditioned, 248
gmresr, 247

gmresrPreconditioner, 249
jacvec, 249
kmsPreconditioner, 249
krylov_method, 242
krylovMultiSpace, 249
MIN_TOL, 242
NumericalJacobian, 250
operatorTranspose, 250
pcg, 251
picard, 252
pjfnk, 252
update_arnoldi_solution, 253
lark_picard_step
finch.h, 230
level
KMS_DATA, 131
SCOPSOWL_DATA, 174
lin_tol_abs
PJFNK_DATA, 167
lin_tol_rel
PJFNK_DATA, 167
LineSearch
PJFNK_DATA, 168
linear_solver
PJFNK_DATA, 167
LinearDrivingForce, 132
_coef, 134
_driving_value, 134
_gaining, 134
_var, 134
computeQpJacobian, 133
computeQpResidual, 133
LinearDrivingForce, 133
LinearDrivingForce, 133
LinearDrivingForce.h, 253
validParams< LinearDrivingForce >, 254
lnKo
magpie.h, 258
lnact_mSPD
magpie.h, 261
lowerHessenberg2Triangular
Matrix, 149
lowerHessenbergSolve
Matrix, 149
lowerTriangularSolve
Matrix, 149

m
GSTA_DATA, 129
M_PI
macaw.h, 255
MAGPIE
magpie.h, 261
MAGPIE_Adsorption, 134
_index, 135
_magpie_dat, 135
computeValue, 135
MAGPIE_Adsorption, 135
MAGPIE_Adsorption, 135
MAGPIE_Adsorption.h, 263
validParams< MAGPIE_Adsorption >, 264
MAGPIE_AdsorptionHeat, 135
_index, 137
_magpie_dat, 137
_solid_conc, 137
computeValue, 136
MAGPIE_AdsorptionHeat, 136
MAGPIE_AdsorptionHeat, 136
MAGPIE_AdsorptionHeat.h, 264
MAGPIE_DATA, 137
gpast_dat, 137
gsta_dat, 137
mspd_dat, 137
sys_dat, 137
MAGPIE_Perturbation, 138
_index, 139
_magpie_dat, 139
computeValue, 139
MAGPIE_Perturbation, 138
MAGPIE_Perturbation, 138
MAGPIE_Perturbation.h, 265
validParams< MAGPIE_Perturbation >, 266
ME
FINCH_DATA, 91
mError
error.h, 223
MI
FINCH_DATA, 91
MIN_TOL
lark.h, 242
MIXED_GAS, 154
binary_diffusion, 155
char_length, 155
CheckMolefractions, 155
gas_temperature, 155
kinematic_viscosity, 155
molefraction, 155
N, 156
Reynolds, 156
species_dat, 156
total_density, 156
total_dyn_vis, 156
total_molecular_weight, 156
total_pressure, 156
total_specific_heat, 156
velocity, 156
mSPD_DATA, 157
eMax, 157
eta, 157
gama, 157
s, 157
v, 157
macaw.h, 254
M_PI, 255
magpie.h, 255
A, 258
DBL_EPSILON, 258

dq_dp, 259
eMax, 259
eval_GPAST, 259
eval_eta, 259
eval_po, 260
eval_po_PI, 260
eval_po_qo, 260
grad_mSPD, 260
He, 258
initialGuess_mSPD, 261
kB, 258
lnKo, 258
lnact_mSPD, 261
MAGPIE, 261
Na, 258
PI, 262
Po, 258
q_p, 262
qT, 263
qo, 262
Qst, 263
R, 258
shapeFactor, 258
V, 259
Z, 259
magpie_reverse_error
error.h, 224
magpie_dat
SCOPSOWL_DATA, 175
SKUA_DATA, 182
MagpieAdsorbateProperties, 139
_enthalpy_1, 141
_enthalpy_2, 141
_enthalpy_3, 141
_enthalpy_4, 141
_enthalpy_5, 141
_enthalpy_6, 141
_entropy_1, 141
_entropy_2, 142
_entropy_3, 142
_entropy_4, 142
_entropy_5, 142
_entropy_6, 142
_gas_conc, 142
_gas_conc_old, 142
_index, 142
_magpie_dat, 142
_max_capacity, 142
_molar_volume, 143
_num_sites, 143
_temperature, 143
_total_pressure, 143
computeQpProperties, 141
MagpieAdsorbateProperties, 141
MagpieAdsorbateProperties, 141
MagpieAdsorbateProperties.h, 266
validParams< MagpieAdsorbateProperties >, 267
Matrix
~Matrix, 146
adjoint, 146
cofactor, 146
columnExtend, 146
columnExtract, 147
columnProjection, 147
columnReplace, 147
columnShrink, 147
columnVectorFill, 147
columns, 147
ConstantICFill, 147
Data, 153
determinate, 147
diagonalSolve, 148
dirichletBCFill, 148
Display, 148
edit, 148
inner_product, 148
IntegralAvg, 148
IntegralTotal, 149
inverse, 149
ladshawSolve, 149
lowerHessenberg2Triangular, 149
lowerHessenbergSolve, 149
lowerTriangularSolve, 149
Matrix, 146
naturalLaplacian3D, 150
norm, 150
num_cols, 154
num_rows, 154
operator*, 150
operator(), 150
operator+, 150
operator-, 150
operator/, 150
operator=, 151
rowExtend, 151
rowExtract, 151
rowReplace, 151
rowShrink, 151
rows, 151
set_size, 151
SolnTransform, 151
sphericalAvg, 151
sphericalBCFill, 152
sum, 152
transpose, 152
transpose_multiply, 152
tridiagonalFill, 152
tridiagonalSolve, 152
tridiagonalVectorFill, 153
upperHessenberg2Triangular, 153
upperHessenbergSolve, 153
upperTriangularSolve, 153
zeros, 153
Matrix< T >, 143
matrix_too_small
error.h, 224

matvec
GMRESR_DATA, 122
KMS_DATA, 131
matvec_mis_match
error.h, 224
matvec_data
GMRESR_DATA, 122
KMS_DATA, 131
max
finch.h, 230
max_iter
FINCH_DATA, 91
max_level
KMS_DATA, 131
max_norm
SYSTEM_DATA, 187
maxit
BiCGSTAB_DATA, 29
CGS_DATA, 33
GCR_DATA, 116
GMRESLP_DATA, 119
GMRESRP_DATA, 125
KMS_DATA, 131
PCG_DATA, 161
PICARD_DATA, 163
min
finch.h, 230
minmod
finch.h, 230
minmod_discretization
finch.h, 230
missing_information
error.h, 225
molecular_diffusion
PURE_GAS, 170
molecular_weight
PURE_GAS, 170
molefraction
MIXED_GAS, 155
molefractionCheck
skua.h, 277
mspd_dat
MAGPIE_DATA, 137
Mu
egret.h, 221

N
GMRESR_DATA, 123
MIXED_GAS, 156
SYSTEM_DATA, 187
NE
FINCH_DATA, 91
NI
FINCH_DATA, 91
NL_Output
PJFNK_DATA, 168
NUM_JAC_DATA, 158
dxj, 158
eps, 158
Fx, 158
Fxp, 158
Na
magpie.h, 258
naturalLaplacian3D
Matrix, 150
negative_mass
error.h, 224
negative_time
error.h, 224
nl_bestres
PJFNK_DATA, 168
nl_iter
PJFNK_DATA, 168
nl_maxit
PJFNK_DATA, 168
nl_method
FINCH_DATA, 91
nl_picard
finch.h, 230
nl_relres
PJFNK_DATA, 168
nl_res
PJFNK_DATA, 168
nl_res_base
PJFNK_DATA, 168
nl_tol_abs
PJFNK_DATA, 168
nl_tol_rel
PJFNK_DATA, 168
no_diffusion
error.h, 224
non_real_edge
error.h, 225
non_square_matrix
error.h, 224
NonLinear
SCOPSOWL_DATA, 175
SKUA_DATA, 182
norm
Matrix, 150
normFkp1
BACKTRACK_DATA, 19
NormTrack
FINCH_DATA, 91
not_a_token
error.h, 225
Nu
egret.h, 221
nullptr_error
error.h, 225
nullptr_func
error.h, 224
num_cols
Matrix, 154
num_rows
Matrix, 154
NumericalJacobian

lark.h, 250

OE
FINCH_DATA, 91
OI
FINCH_DATA, 92
OPTRANS_DATA, 159
Ai, 159
Ii, 159
omega
BiCGSTAB_DATA, 29
omega_old
BiCGSTAB_DATA, 29
operator∗
Matrix, 150
operator()
Matrix, 150
operator+
Matrix, 150
operator-
Matrix, 150
operator/
Matrix, 150
operator=
Matrix, 151
operatorTranspose
lark.h, 250
opt_no_support
error.h, 224
ortho_check_fail
error.h, 224
ospre_discretization
finch.h, 230
out_of_bounds
error.h, 224
outer_abstol
KMS_DATA, 131
outer_iter
KMS_DATA, 131
outer_reltol
KMS_DATA, 131
Output
ARNOLDI_DATA, 17
BiCGSTAB_DATA, 30
CGS_DATA, 33
GCR_DATA, 116
GMRESLP_DATA, 119
GMRESRP_DATA, 125
PCG_DATA, 161
PICARD_DATA, 163
SYSTEM_DATA, 187
Output_in
KMS_DATA, 132
Output_out
KMS_DATA, 132
OutputFile
SCOPSOWL_DATA, 175
SKUA_DATA, 182

p
BiCGSTAB_DATA, 30
CGS_DATA, 33
PCG_DATA, 161
PCG
lark.h, 243
PCG_DATA, 159
alpha, 160
Ap, 160
bestres, 160
bestx, 161
beta, 161
iter, 161
maxit, 161
Output, 161
p, 161
r, 161
r_old, 161
relres, 161
relres_base, 161
res, 161
tol_abs, 162
tol_rel, 162
x, 162
z, 162
z_old, 162
PE3
egret.h, 221
PI
magpie.h, 262
SYSTEM_DATA, 187
PICARD_DATA, 162
bestres, 163
bestx, 163
iter, 163
maxit, 163
Output, 163
r, 163
relres, 163
relres_base, 164
res, 164
tol_abs, 164
tol_rel, 164
x0, 164
PIo
GPAST_DATA, 128
PJFNK_DATA, 164
backtrack_dat, 166
bestx, 166
bicgstab_dat, 166
Bounce, 166
cgs_dat, 166
eps, 166
F, 166
fun_call, 166
funeval, 167
Fv, 167
gcr_dat, 167

gmreslp_dat, 167
gmresr_dat, 167
gmresrp_dat, 167
L_Output, 167
l_iter, 167
lin_tol_abs, 167
lin_tol_rel, 167
LineSearch, 168
linear_solver, 167
NL_Output, 168
nl_bestres, 168
nl_iter, 168
nl_maxit, 168
nl_relres, 168
nl_res, 168
nl_res_base, 168
nl_tol_abs, 168
nl_tol_rel, 168
pcg_dat, 168
precon, 169
precon_data, 169
res_data, 169
v, 169
x, 169
PSI
egret.h, 221
PT
SYSTEM_DATA, 187
PURE_GAS, 169
density, 170
dynamic_viscosity, 170
molecular_diffusion, 170
molecular_weight, 170
Schmidt, 170
specific_heat, 170
Sutherland_Const, 170
Sutherland_Temp, 170
Sutherland_Viscosity, 171
Par
SYSTEM_DATA, 187
param_dat
SCOPSOWL_DATA, 175
SKUA_DATA, 182
param_data
FINCH_DATA, 92
pcg
lark.h, 251
pcg_dat
PJFNK_DATA, 168
pellet_density
SCOPSOWL_DATA, 175
pellet_radius
SCOPSOWL_DATA, 175
SKUA_DATA, 182
pi
SYSTEM_DATA, 187
picard
lark.h, 252
picard_dat
FINCH_DATA, 92
pjfnk
lark.h, 252
pjfnk_dat
FINCH_DATA, 92
Po
egret.h, 221
magpie.h, 258
po
GPAST_DATA, 128
poi
GPAST_DATA, 128
pore_diffusion
SCOPSOWL_PARAM_DATA, 178
precon
PJFNK_DATA, 169
precon_data
PJFNK_DATA, 169
pres
FINCH_DATA, 92
present
GPAST_DATA, 128
Print2Console
SCOPSOWL_DATA, 175
SKUA_DATA, 182
Print2File
SCOPSOWL_DATA, 175
SKUA_DATA, 182
print2file_SCOPSOWL_header
scopsowl.h, 271
print2file_SCOPSOWL_result_new
scopsowl.h, 271
print2file_SCOPSOWL_result_old
scopsowl.h, 271
print2file_SCOPSOWL_time_header
scopsowl.h, 271
print2file_SKUA_header
skua.h, 277
print2file_SKUA_results_new
skua.h, 277
print2file_SKUA_results_old
skua.h, 277
print2file_SKUA_time_header
skua.h, 277
print2file_dim_header
finch.h, 230
print2file_newline
finch.h, 230
print2file_result_new
finch.h, 231
print2file_result_old
finch.h, 231
print2file_species_header
scopsowl.h, 271
skua.h, 277
print2file_tab
finch.h, 231

print2file_time_header
finch.h, 231
Pstd
egret.h, 221

q
GPAST_DATA, 128
q_p
magpie.h, 262
qAvg
SCOPSOWL_PARAM_DATA, 178
qAvg_old
SCOPSOWL_PARAM_DATA, 178
qIntegralAvg
SCOPSOWL_PARAM_DATA, 178
qIntegralAvg_old
SCOPSOWL_PARAM_DATA, 179
qT
magpie.h, 263
SYSTEM_DATA, 187
qTn
SKUA_DATA, 182
qTnp1
SKUA_DATA, 183
qmax
GSTA_DATA, 129
qo
GPAST_DATA, 128
magpie.h, 262
SCOPSOWL_PARAM_DATA, 179
Qst
magpie.h, 263
SCOPSOWL_PARAM_DATA, 179
Qst_old
SCOPSOWL_PARAM_DATA, 179
QstAvg
SCOPSOWL_PARAM_DATA, 179
QstAvg_old
SCOPSOWL_PARAM_DATA, 179
Qstn
SKUA_PARAM, 184
Qstnp1
SKUA_PARAM, 184
Qsto
SCOPSOWL_PARAM_DATA, 179

R
magpie.h, 258
r
BiCGSTAB_DATA, 30
CGS_DATA, 33
GCR_DATA, 116
GMRESLP_DATA, 119
GMRESRP_DATA, 125
PCG_DATA, 161
PICARD_DATA, 163
r0
BiCGSTAB_DATA, 30
CGS_DATA, 33
r_old
PCG_DATA, 161
RE3
egret.h, 221
RIC
FINCH_DATA, 92
ReNum
egret.h, 221
read_error
error.h, 225
Recover
SYSTEM_DATA, 187
ref_diffusion
SCOPSOWL_PARAM_DATA, 179
SKUA_PARAM, 184
ref_pressure
SCOPSOWL_PARAM_DATA, 179
SKUA_PARAM, 185
ref_temperature
SCOPSOWL_PARAM_DATA, 179
SKUA_PARAM, 185
registerApps
DgospreyApp, 82
registerObjects
DgospreyApp, 82
relres
BiCGSTAB_DATA, 30
CGS_DATA, 33
GCR_DATA, 116
GMRESLP_DATA, 119
GMRESRP_DATA, 125
PCG_DATA, 161
PICARD_DATA, 163
relres_base
BiCGSTAB_DATA, 30
CGS_DATA, 34
GCR_DATA, 117
GMRESLP_DATA, 119
GMRESRP_DATA, 126
PCG_DATA, 161
PICARD_DATA, 164
res
BiCGSTAB_DATA, 30
CGS_DATA, 34
FINCH_DATA, 92
GCR_DATA, 117
GMRESLP_DATA, 119
GMRESRP_DATA, 126
PCG_DATA, 161
PICARD_DATA, 164
res_data
PJFNK_DATA, 169
resettime
FINCH_DATA, 92
restart
GCR_DATA, 117
GMRESLP_DATA, 119
GMRESRP_DATA, 126

KMS_DATA, 132
Reynolds
MIXED_GAS, 156
rho
BACKTRACK_DATA, 19
BiCGSTAB_DATA, 30
CGS_DATA, 34
rho_old
BiCGSTAB_DATA, 30
Rn
FINCH_DATA, 92
Rnp1
FINCH_DATA, 92
Ro
FINCH_DATA, 92
rowExtend
Matrix, 151
rowExtract
Matrix, 151
rowReplace
Matrix, 151
rowShrink
Matrix, 151
rows
Matrix, 151
Rstd
egret.h, 221
rxn_rate_error
error.h, 225

s
BiCGSTAB_DATA, 30
FINCH_DATA, 93
mSPD_DATA, 157
SCOPSOWL
scopsowl.h, 271
SCOPSOWL_DATA, 171
binder_fraction, 173
binder_poresize, 173
binder_porosity, 173
char_macro, 173
char_micro, 173
coord_macro, 173
coord_micro, 173
crystal_radius, 173
DirichletBC, 173
eval_ads, 174
eval_diff, 174
eval_kf, 174
eval_retard, 174
eval_surfDiff, 174
finch_dat, 174
gas_dat, 174
gas_temperature, 174
gas_velocity, 174
Heterogeneous, 174
level, 174
magpie_dat, 175
NonLinear, 175
OutputFile, 175
param_dat, 175
pellet_density, 175
pellet_radius, 175
Print2Console, 175
Print2File, 175
sim_time, 175
skua_dat, 175
SurfDiff, 175
t, 176
t_counter, 176
t_old, 176
t_print, 176
tempy, 176
total_pressure, 176
total_steps, 176
user_data, 176
y, 176
SCOPSOWL_Executioner
scopsowl.h, 271
SCOPSOWL_HPP_
scopsowl.h, 269
SCOPSOWL_PARAM_DATA, 177
Adsorbable, 178
affinity, 178
qAvg, 178
qo, 179
Qst, 179
QstAvg, 179
Qsto, 179
speciesName, 179
SCOPSOWL_postprocesses
scopsowl.h, 272
SCOPSOWL_preprocesses
scopsowl.h, 272
SCOPSOWL_reset
scopsowl.h, 272
SKUA
skua.h, 278
SKUA_DATA, 180
char_measure, 181
coord, 181
DirichletBC, 181
eval_diff, 181
eval_kf, 181
finch_dat, 182
gas_dat, 182
gas_velocity, 182
magpie_dat, 182
NonLinear, 182
OutputFile, 182
param_dat, 182
pellet_radius, 182
Print2Console, 182
Print2File, 182
qTn, 182
qTnp1, 183
sim_time, 183

t, 183
t_counter, 183
t_old, 183
t_print, 183
total_steps, 183
user_data, 183
y, 183
SKUA_Executioner
skua.h, 278
SKUA_HPP_
skua.h, 276
SKUA_PARAM, 184
activation_energy, 184
Adsorbable, 184
affinity, 184
film_transfer, 184
Qstn, 184
Qstnp1, 184
ref_diffusion, 184
ref_pressure, 185
ref_temperature, 185
speciesName, 185
xIC, 185
xn, 185
xnp1, 185
y_eff, 185
SKUA_postprocesses
skua.h, 278
SKUA_preprocesses
skua.h, 279
SKUA_reset
skua.h, 279
SYSTEM_DATA, 185
As, 186
avg_norm, 186
Carrier, 186
I, 186
Ideal, 186
J, 187
K, 187
max_norm, 187
N, 187
Output, 187
PI, 187
PT, 187
Par, 187
pi, 187
qT, 187
Recover, 187
Sys, 188
T, 188
total_eval, 188
ScNum
egret.h, 221
scenario_fail
error.h, 224
Schmidt
PURE_GAS, 170
scopsowl.h, 267
avgDp, 269
const_filmMassTransfer, 269
const_pore_diffusion, 269
default_adsorption, 270
default_effective_diffusion, 270
default_filmMassTransfer, 270
default_pore_diffusion, 270
default_retardation, 270
default_surf_diffusion, 271
Dk, 269
Dp, 269
print2file_SCOPSOWL_header, 271
print2file_SCOPSOWL_result_new, 271
print2file_SCOPSOWL_result_old, 271
print2file_SCOPSOWL_time_header, 271
print2file_species_header, 271
SCOPSOWL, 271
SCOPSOWL_Executioner, 271
SCOPSOWL_HPP_, 269
SCOPSOWL_postprocesses, 272
SCOPSOWL_preprocesses, 272
SCOPSOWL_reset, 272
set_SCOPSOWL_ICs, 272
set_SCOPSOWL_params, 272
set_SCOPSOWL_timestep, 273
setup_SCOPSOWL_DATA, 273
set_SCOPSOWL_ICs
scopsowl.h, 272
set_SCOPSOWL_params
scopsowl.h, 272
set_SCOPSOWL_timestep
scopsowl.h, 273
set_SKUA_ICs
skua.h, 277
set_SKUA_params
skua.h, 277
set_SKUA_timestep
skua.h, 277
set_size
Matrix, 151
set_variables
egret.h, 222
setbcs
FINCH_DATA, 93
setic
FINCH_DATA, 93
setparams
FINCH_DATA, 93
setpostprocess
FINCH_DATA, 93
setpreprocess
FINCH_DATA, 93
settime
FINCH_DATA, 93
setup_FINCH_DATA
finch.h, 231
setup_SCOPSOWL_DATA

scopsowl.h, 273
setup_SKUA_DATA
skua.h, 278
shapeFactor
magpie.h, 258
sigma
CGS_DATA, 34
sim_time
SCOPSOWL_DATA, 175
SKUA_DATA, 183
simple_darken_Dc
skua.h, 278
simulation_fail
error.h, 224
singular_matrix
error.h, 224
skua.h, 273
const_Dc, 276
const_kf, 276
D_c, 275
D_inf, 275
D_o, 275
default_Dc, 276
default_kf, 276
empirical_kf, 277
molefractionCheck, 277
print2file_SKUA_header, 277
print2file_SKUA_results_new, 277
print2file_SKUA_results_old, 277
print2file_SKUA_time_header, 277
print2file_species_header, 277
SKUA, 278
SKUA_Executioner, 278
SKUA_HPP_, 276
SKUA_postprocesses, 278
SKUA_preprocesses, 279
SKUA_reset, 279
set_SKUA_ICs, 277
set_SKUA_params, 277
set_SKUA_timestep, 277
setup_SKUA_DATA, 278
simple_darken_Dc, 278
theoretical_darken_Dc, 279
skua_dat
SCOPSOWL_DATA, 175
Sn
FINCH_DATA, 93
Snp1
FINCH_DATA, 93
SolnTransform
Matrix, 151
solve
FINCH_DATA, 93
species_dat
MIXED_GAS, 156
speciesName
SCOPSOWL_PARAM_DATA, 179
SKUA_PARAM, 185
specific_heat
PURE_GAS, 170
Spherical
finch.h, 228
sphericalAvg
Matrix, 151
sphericalBCFill
Matrix, 152
SteadyState
FINCH_DATA, 93
steps
GMRESLP_DATA, 119
string_parse_error
error.h, 225
sum
ARNOLDI_DATA, 17
GMRESRP_DATA, 126
Matrix, 152
SurfDiff
SCOPSOWL_DATA, 175
Sutherland_Const
PURE_GAS, 170
Sutherland_Temp
PURE_GAS, 170
Sutherland_Viscosity
PURE_GAS, 171
Sys
SYSTEM_DATA, 188
sys_dat
MAGPIE_DATA, 137

T
FINCH_DATA, 94
SYSTEM_DATA, 188
t
BiCGSTAB_DATA, 30
FINCH_DATA, 94
SCOPSOWL_DATA, 176
SKUA_DATA, 183
t_counter
SCOPSOWL_DATA, 176
SKUA_DATA, 183
t_old
FINCH_DATA, 94
SCOPSOWL_DATA, 176
SKUA_DATA, 183
t_print
SCOPSOWL_DATA, 176
SKUA_DATA, 183
tempy
SCOPSOWL_DATA, 176
tensor_out_of_bounds
error.h, 225
term_precon
GMRESR_DATA, 123
KMS_DATA, 132
terminal_precon
GMRESR_DATA, 123
KMS_DATA, 132

theoretical_darken_Dc
skua.h, 279
tol_abs
BiCGSTAB_DATA, 30
CGS_DATA, 34
FINCH_DATA, 94
GCR_DATA, 117
GMRESLP_DATA, 120
GMRESRP_DATA, 126
PCG_DATA, 162
PICARD_DATA, 164
tol_rel
BiCGSTAB_DATA, 31
CGS_DATA, 34
FINCH_DATA, 94
GCR_DATA, 117
GMRESLP_DATA, 120
GMRESRP_DATA, 126
PCG_DATA, 162
PICARD_DATA, 164
total_density
MIXED_GAS, 156
total_dyn_vis
MIXED_GAS, 156
total_eval
SYSTEM_DATA, 188
total_iter
FINCH_DATA, 94
GCR_DATA, 117
GMRESR_DATA, 123
KMS_DATA, 132
total_molecular_weight
MIXED_GAS, 156
total_pressure
MIXED_GAS, 156
SCOPSOWL_DATA, 176
total_specific_heat
MIXED_GAS, 156
total_steps
SCOPSOWL_DATA, 176
SKUA_DATA, 183
TotalColumnPressure, 188
_gas_conc, 189
_index, 189
_temperature, 189
computeValue, 189
TotalColumnPressure, 189
TotalColumnPressure, 189
TotalColumnPressure.h, 279
validParams< TotalColumnPressure >, 280
TotalPressureIC, 189
_PT_IC, 190
TotalPressureIC, 190
TotalPressureIC, 190
value, 190
TotalPressureIC.h, 280
validParams< TotalPressureIC >, 281
transpose
Matrix, 152
transpose_dat
GCR_DATA, 117
transpose_multiply
Matrix, 152
tridiagonalFill
Matrix, 152
tridiagonalSolve
Matrix, 152
tridiagonalVectorFill
Matrix, 153

u
CGS_DATA, 34
GCR_DATA, 117
u_star
FINCH_DATA, 94
u_temp
GCR_DATA, 117
uAverage
finch.h, 231
uAvg
FINCH_DATA, 94
uAvg_old
FINCH_DATA, 94
uIC
FINCH_DATA, 94
uT
FINCH_DATA, 95
uT_old
FINCH_DATA, 95
uTotal
finch.h, 231
ubest
FINCH_DATA, 94
un
FINCH_DATA, 95
unm1
FINCH_DATA, 95
unp1
FINCH_DATA, 95
unregistered_name
error.h, 225
unstable_matrix
error.h, 224
uo
FINCH_DATA, 95
Update
FINCH_DATA, 95
update_arnoldi_solution
lark.h, 253
upperHessenberg2Triangular
Matrix, 153
upperHessenbergSolve
Matrix, 153
upperTriangularSolve
Matrix, 153
user_data
SCOPSOWL_DATA, 176

SKUA_DATA, 183
uz_l_E
FINCH_DATA, 95
uz_l_I
FINCH_DATA, 95
uz_lm1_E
FINCH_DATA, 95
uz_lm1_I
FINCH_DATA, 95
uz_lp1_E
FINCH_DATA, 95
uz_lp1_I
FINCH_DATA, 96

V
magpie.h, 259
v
ARNOLDI_DATA, 17
BiCGSTAB_DATA, 31
CGS_DATA, 34
GMRESRP_DATA, 126
mSPD_DATA, 157
PJFNK_DATA, 169
vIC
FINCH_DATA, 96
validParams< AdsorbentProperties >
AdsorbentProperties.h, 195
validParams< AdsorptionHeatAccumulation >
AdsorptionHeatAccumulation.h, 196
validParams< AdsorptionMassTransfer >
AdsorptionMassTransfer.h, 197
validParams< BedHeatAccumulation >
BedHeatAccumulation.h, 198
validParams< BedMassAccumulation >
BedMassAccumulation.h, 199
validParams< BedProperties >
BedProperties.h, 199
validParams< BedWallHeatTransfer >
BedWallHeatTransfer.h, 200
validParams< ColumnTemperatureIC >
ColumnTemperatureIC.h, 201
validParams< ConcentrationIC >
ConcentrationIC.h, 202
validParams< CoupledLDF >
CoupledLDF.h, 203
validParams< DGAdvection >
DGAdvection.h, 204
validParams< DGAnisotropicDiffusion >
DGAnisotropicDiffusion.h, 205
validParams< DGColumnHeatAdvection >
DGColumnHeatAdvection.h, 206
validParams< DGColumnHeatDispersion >
DGColumnHeatDispersion.h, 207
validParams< DGColumnMassAdvection >
DGColumnMassAdvection.h, 208
validParams< DGColumnMassDispersion >
DGColumnMassDispersion.h, 210
validParams< DGColumnWallHeatFluxBC >
DGColumnWallHeatFluxBC.h, 210
validParams< DGColumnWallHeatFluxLimitedBC >
DGColumnWallHeatFluxLimitedBC.h, 211
validParams< DGFluxBC >
DGFluxBC.h, 212
validParams< DGFluxLimitedBC >
DGFluxLimitedBC.h, 213
validParams< DGHeatFluxBC >
DGHeatFluxBC.h, 214
validParams< DGHeatFluxLimitedBC >
DGHeatFluxLimitedBC.h, 215
validParams< DGMassFluxBC >
DGMassFluxBC.h, 216
validParams< DGMassFluxLimitedBC >
DGMassFluxLimitedBC.h, 217
validParams< DgospreyApp >
DgospreyApp.h, 218
validParams< FlowProperties >
FlowProperties.h, 234
validParams< GAdvection >
GAdvection.h, 234
validParams< GAnisotropicDiffusion >
GAnisotropicDiffusion.h, 235
validParams< GColumnHeatAdvection >
GColumnHeatAdvection.h, 236
validParams< GColumnHeatDispersion >
GColumnHeatDispersion.h, 237
validParams< GColumnMassAdvection >
GColumnMassAdvection.h, 238
validParams< GColumnMassDispersion >
GColumnMassDispersion.h, 239
validParams< LinearDrivingForce >
LinearDrivingForce.h, 254
validParams< MAGPIE_Adsorption >
MAGPIE_Adsorption.h, 264
validParams< MAGPIE_AdsorptionHeat >
MAGPIE_AdsorptionHeat.h, 265
validParams< MAGPIE_Perturbation >
MAGPIE_Perturbation.h, 266
validParams< MagpieAdsorbateProperties >
MagpieAdsorbateProperties.h, 267
validParams< TotalColumnPressure >
TotalColumnPressure.h, 280
validParams< TotalPressureIC >
TotalPressureIC.h, 281
validParams< WallAmbientHeatTransfer >
WallAmbientHeatTransfer.h, 282
validParams< WallHeatAccumulation >
WallHeatAccumulation.h, 283
value
ColumnTemperatureIC, 36
ConcentrationIC, 37
TotalPressureIC, 190
vanAlbada_discretization
finch.h, 232
vector_out_of_bounds
error.h, 225
velocity
MIXED_GAS, 156

Vk
ARNOLDI_DATA, 17
GMRESRP_DATA, 126
vn
FINCH_DATA, 96
vnp1
FINCH_DATA, 96
vo
FINCH_DATA, 96

w
ARNOLDI_DATA, 17
CGS_DATA, 34
GMRESRP_DATA, 126
WallAmbientHeatTransfer, 191
_ambient_temp, 192
_inner_dia, 192
_outer_dia, 192
_wall_exterior_transfer_coeff, 192
computeQpJacobian, 192
computeQpResidual, 192
WallAmbientHeatTransfer, 192
WallAmbientHeatTransfer, 192
WallAmbientHeatTransfer.h, 281
validParams< WallAmbientHeatTransfer >, 282
WallHeatAccumulation, 192
_wall_density, 194
_wall_heat_capacity, 194
computeQpJacobian, 193
computeQpResidual, 193
WallHeatAccumulation, 193
WallHeatAccumulation, 193
WallHeatAccumulation.h, 282
validParams< WallHeatAccumulation >, 283

x
BiCGSTAB_DATA, 31
CGS_DATA, 34
GCR_DATA, 117
GMRESLP_DATA, 120
GMRESRP_DATA, 126
GPAST_DATA, 128
PCG_DATA, 162
PJFNK_DATA, 169
x0
PICARD_DATA, 164
xIC
SCOPSOWL_PARAM_DATA, 180
SKUA_PARAM, 185
xk
BACKTRACK_DATA, 19
xn
SKUA_PARAM, 185
xnp1
SKUA_PARAM, 185

y
BiCGSTAB_DATA, 31
GMRESRP_DATA, 126
GPAST_DATA, 128
SCOPSOWL_DATA, 176
SKUA_DATA, 183
y_eff
SKUA_PARAM, 185
yk
ARNOLDI_DATA, 17

Z
magpie.h, 259
z
BiCGSTAB_DATA, 31
CGS_DATA, 34
PCG_DATA, 162
z_old
PCG_DATA, 162
zero_vector
error.h, 225
zeros
Matrix, 153
Zk
GMRESRP_DATA, 127