DGOSPREY

Version 0.0 beta

Generated by Doxygen 1.8.3.1

Mon Feb 15 2016 14:46:10

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 4

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 12

5.1.2 Constructor & Destructor Documentation 12

5.1.3 Member Function Documentation 12

5.1.4 Member Data Documentation 12

5.2 AdsorptionHeatAccumulation Class Reference 15

5.2.1 Detailed Description 15

5.2.2 Constructor & Destructor Documentation 15

5.2.3 Member Function Documentation 16

5.2.4 Member Data Documentation 16

5.3 AdsorptionMassTransfer Class Reference 16

5.3.1 Detailed Description 17

5.3.2 Constructor & Destructor Documentation 17

5.3.3 Member Function Documentation 17

5.3.4 Member Data Documentation 18

5.4 ARNOLDI_DATA Struct Reference 18

5.4.1 Detailed Description 19

5.4.2 Member Data Documentation 19

5.5 Aux_LDF Class Reference 20

5.5.1 Detailed Description 21

5.5.2 Constructor & Destructor Documentation 21

5.5.3 Member Function Documentation 21

5.5.4 Member Data Documentation 21

5.6 BACKTRACK_DATA Struct Reference 21

5.6.1 Detailed Description 22

5.6.2 Member Data Documentation 22

5.7 BedHeatAccumulation Class Reference . . . 23
5.7.1 Detailed Description . . . 23
5.7.2 Constructor & Destructor Documentation . . . 24
5.7.3 Member Function Documentation . . . 24
5.7.4 Member Data Documentation . . . 24
5.8 BedMassAccumulation Class Reference . . . 24
5.8.1 Detailed Description . . . 25
5.8.2 Constructor & Destructor Documentation . . . 25
5.8.3 Member Function Documentation . . . 25
5.8.4 Member Data Documentation . . . 25
5.9 BedProperties Class Reference . . . 26
5.9.1 Detailed Description . . . 27
5.9.2 Constructor & Destructor Documentation . . . 27
5.9.3 Member Function Documentation . . . 27
5.9.4 Member Data Documentation . . . 27
5.10 BedWallHeatTransfer Class Reference . . . 29
5.10.1 Detailed Description . . . 30
5.10.2 Constructor & Destructor Documentation . . . 30
5.10.3 Member Function Documentation . . . 30
5.10.4 Member Data Documentation . . . 31
5.11 BiCGSTAB_DATA Struct Reference . . . 31
5.11.1 Detailed Description . . . 32
5.11.2 Member Data Documentation . . . 32
5.12 CGS_DATA Struct Reference . . . 35
5.12.1 Detailed Description . . . 36
5.12.2 Member Data Documentation . . . 36
5.13 ColumnTemperatureIC Class Reference . . . 38
5.13.1 Detailed Description . . . 39
5.13.2 Constructor & Destructor Documentation . . . 39
5.13.3 Member Function Documentation . . . 39
5.13.4 Member Data Documentation . . . 39
5.14 ConcentrationIC Class Reference . . . 40
5.14.1 Detailed Description . . . 40
5.14.2 Constructor & Destructor Documentation . . . 40
5.14.3 Member Function Documentation . . . 41
5.14.4 Member Data Documentation . . . 41
5.15 CoupledLDF Class Reference . . . 41
5.15.1 Detailed Description . . . 42
5.15.2 Constructor & Destructor Documentation . . . 42
5.15.3 Member Function Documentation . . . 42

5.15.4 Member Data Documentation 43
5.16 DGAdvection Class Reference 43
5.16.1 Detailed Description 44
5.16.2 Constructor & Destructor Documentation 44
5.16.3 Member Function Documentation 45
5.16.4 Member Data Documentation 45
5.17 DGAnisotropicDiffusion Class Reference 45
5.17.1 Detailed Description 46
5.17.2 Constructor & Destructor Documentation 47
5.17.3 Member Function Documentation 47
5.17.4 Member Data Documentation 47
5.18 DGColumnHeatAdvection Class Reference 48
5.18.1 Detailed Description 49
5.18.2 Constructor & Destructor Documentation 49
5.18.3 Member Function Documentation 49
5.18.4 Member Data Documentation 50
5.19 DGColumnHeatDispersion Class Reference 50
5.19.1 Detailed Description 51
5.19.2 Constructor & Destructor Documentation 52
5.19.3 Member Function Documentation 52
5.19.4 Member Data Documentation 52
5.20 DGColumnMassAdvection Class Reference 53
5.20.1 Detailed Description 54
5.20.2 Constructor & Destructor Documentation 54
5.20.3 Member Function Documentation 54
5.20.4 Member Data Documentation 55
5.21 DGColumnMassDispersion Class Reference 55
5.21.1 Detailed Description 56
5.21.2 Constructor & Destructor Documentation 56
5.21.3 Member Function Documentation 57
5.21.4 Member Data Documentation 57
5.22 DGColumnWallHeatFluxBC Class Reference 58
5.22.1 Detailed Description 59
5.22.2 Constructor & Destructor Documentation 59
5.22.3 Member Function Documentation 60
5.22.4 Member Data Documentation 60
5.23 DGColumnWallHeatFluxLimitedBC Class Reference 61
5.23.1 Detailed Description 63
5.23.2 Constructor & Destructor Documentation 63
5.23.3 Member Function Documentation 63

5.23.4 Member Data Documentation 63
5.24 DGFluxBC Class Reference 65
5.24.1 Detailed Description 66
5.24.2 Constructor & Destructor Documentation 66
5.24.3 Member Function Documentation 66
5.24.4 Member Data Documentation 66
5.25 DGFluxLimitedBC Class Reference 67
5.25.1 Detailed Description 68
5.25.2 Constructor & Destructor Documentation 69
5.25.3 Member Function Documentation 69
5.25.4 Member Data Documentation 69
5.26 DGHeatFluxBC Class Reference 70
5.26.1 Detailed Description 72
5.26.2 Constructor & Destructor Documentation 72
5.26.3 Member Function Documentation 72
5.26.4 Member Data Documentation 72
5.27 DGHeatFluxLimitedBC Class Reference 74
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 76
5.28 DGMassFluxBC 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 DGMassFluxLimitedBC Class Reference 81
5.29.1 Detailed Description 83
5.29.2 Constructor & Destructor Documentation 83
5.29.3 Member Function Documentation 83
5.29.4 Member Data Documentation 83
5.30 DgospreyApp Class Reference 85
5.30.1 Detailed Description 86
5.30.2 Constructor & Destructor Documentation 86
5.30.3 Member Function Documentation 86
5.31 FINCH_DATA Struct Reference 86
5.31.1 Detailed Description 91
5.31.2 Member Data Documentation 91
5.32 FlowProperties Class Reference 100
5.32.1 Detailed Description 102

5.32.2 Constructor & Destructor Documentation 102
5.32.3 Member Function Documentation 102
5.32.4 Member Data Documentation 103
5.33 GAdvection Class Reference 106
5.33.1 Detailed Description 107
5.33.2 Constructor & Destructor Documentation 107
5.33.3 Member Function Documentation 107
5.33.4 Member Data Documentation 107
5.34 GAnisotropicDiffusion Class Reference 108
5.34.1 Detailed Description 109
5.34.2 Constructor & Destructor Documentation 109
5.34.3 Member Function Documentation 109
5.34.4 Member Data Documentation 109
5.35 GColumnHeatAdvection Class Reference 110
5.35.1 Detailed Description 111
5.35.2 Constructor & Destructor Documentation 111
5.35.3 Member Function Documentation 111
5.35.4 Member Data Documentation 112
5.36 GColumnHeatDispersion Class Reference 112
5.36.1 Detailed Description 113
5.36.2 Constructor & Destructor Documentation 113
5.36.3 Member Function Documentation 114
5.36.4 Member Data Documentation 114
5.37 GColumnMassAdvection Class Reference 115
5.37.1 Detailed Description 116
5.37.2 Constructor & Destructor Documentation 116
5.37.3 Member Function Documentation 116
5.37.4 Member Data Documentation 116
5.38 GColumnMassDispersion Class Reference 117
5.38.1 Detailed Description 118
5.38.2 Constructor & Destructor Documentation 118
5.38.3 Member Function Documentation 118
5.38.4 Member Data Documentation 118
5.39 GCR_DATA Struct Reference 119
5.39.1 Detailed Description 121
5.39.2 Member Data Documentation 121
5.40 GMRESLP_DATA Struct Reference 123
5.40.1 Detailed Description 124
5.40.2 Member Data Documentation 124
5.41 GMRESR_DATA Struct Reference 125

5.41.1 Detailed Description . . . 126
5.41.2 Member Data Documentation . . . 126
5.42 GMRESRP_DATA Struct Reference . . . 128
5.42.1 Detailed Description . . . 130
5.42.2 Member Data Documentation . . . 130
5.43 GPAST_DATA Struct Reference . . . 132
5.43.1 Detailed Description . . . 133
5.43.2 Member Data Documentation . . . 133
5.44 GSTA_DATA Struct Reference . . . 134
5.44.1 Detailed Description . . . 134
5.44.2 Member Data Documentation . . . 134
5.45 KMS_DATA Struct Reference . . . 135
5.45.1 Detailed Description . . . 136
5.45.2 Member Data Documentation . . . 136
5.46 LinearDrivingForce Class Reference . . . 138
5.46.1 Detailed Description . . . 138
5.46.2 Constructor & Destructor Documentation . . . 139
5.46.3 Member Function Documentation . . . 139
5.46.4 Member Data Documentation . . . 139
5.47 MAGPIE_Adsorption Class Reference . . . 139
5.47.1 Detailed Description . . . 140
5.47.2 Constructor & Destructor Documentation . . . 140
5.47.3 Member Function Documentation . . . 140
5.47.4 Member Data Documentation . . . 141
5.48 MAGPIE_AdsorptionHeat Class Reference . . . 141
5.48.1 Detailed Description . . . 142
5.48.2 Constructor & Destructor Documentation . . . 142
5.48.3 Member Function Documentation . . . 142
5.48.4 Member Data Documentation . . . 142
5.49 MAGPIE_ConstLDF_Adsorption Class Reference . . . 142
5.49.1 Detailed Description . . . 143
5.49.2 Constructor & Destructor Documentation . . . 143
5.49.3 Member Function Documentation . . . 144
5.49.4 Member Data Documentation . . . 144
5.50 MAGPIE_ConstLDF_Perturbation Class Reference . . . 144
5.50.1 Detailed Description . . . 145
5.50.2 Constructor & Destructor Documentation . . . 145
5.50.3 Member Function Documentation . . . 145
5.50.4 Member Data Documentation . . . 145
5.51 MAGPIE_DATA Struct Reference . . . 146

5.51.1 Detailed Description 146
5.51.2 Member Data Documentation 146
5.52 MAGPIE_MaterialLDF_Adsorption Class Reference 147
5.52.1 Detailed Description 148
5.52.2 Constructor & Destructor Documentation 148
5.52.3 Member Function Documentation 148
5.52.4 Member Data Documentation 148
5.53 MAGPIE_MaterialLDF_Perturbation Class Reference 149
5.53.1 Detailed Description 151
5.53.2 Constructor & Destructor Documentation 151
5.53.3 Member Function Documentation 151
5.53.4 Member Data Documentation 151
5.54 MAGPIE_Perturbation Class Reference 152
5.54.1 Detailed Description 153
5.54.2 Constructor & Destructor Documentation 153
5.54.3 Member Function Documentation 153
5.54.4 Member Data Documentation 153
5.55 MagpieAdsorbateProperties Class Reference 154
5.55.1 Detailed Description 155
5.55.2 Constructor & Destructor Documentation 155
5.55.3 Member Function Documentation 155
5.55.4 Member Data Documentation 156
5.56 Matrix< T > Class Template Reference 158
5.56.1 Detailed Description 161
5.56.2 Constructor & Destructor Documentation 161
5.56.3 Member Function Documentation 161
5.56.4 Member Data Documentation 168
5.57 MIXED_GAS Struct Reference 169
5.57.1 Detailed Description 170
5.57.2 Member Data Documentation 170
5.58 mSPD_DATA Struct Reference 172
5.58.1 Detailed Description 172
5.58.2 Member Data Documentation 172
5.59 NUM_JAC_DATA Struct Reference 173
5.59.1 Detailed Description 173
5.59.2 Member Data Documentation 173
5.60 OPTRANS_DATA Struct Reference 174
5.60.1 Detailed Description 174
5.60.2 Member Data Documentation 174
5.61 PCG_DATA Struct Reference 174

5.61.1 Detailed Description . . . 175
5.61.2 Member Data Documentation . . . 175
5.62 PICARD_DATA Struct Reference . . . 177
5.62.1 Detailed Description . . . 178
5.62.2 Member Data Documentation . . . 178
5.63 PJFNK_DATA Struct Reference . . . 179
5.63.1 Detailed Description . . . 181
5.63.2 Member Data Documentation . . . 181
5.64 PURE_GAS Struct Reference . . . 184
5.64.1 Detailed Description . . . 185
5.64.2 Member Data Documentation . . . 185
5.65 SCOPSOWL_DATA Struct Reference . . . 186
5.65.1 Detailed Description . . . 188
5.65.2 Member Data Documentation . . . 188
5.66 SCOPSOWL_PARAM_DATA Struct Reference . . . 192
5.66.1 Detailed Description . . . 193
5.66.2 Member Data Documentation . . . 193
5.67 SKUA_DATA Struct Reference . . . 195
5.67.1 Detailed Description . . . 196
5.67.2 Member Data Documentation . . . 196
5.68 SKUA_PARAM Struct Reference . . . 199
5.68.1 Detailed Description . . . 199
5.68.2 Member Data Documentation . . . 199
5.69 SYSTEM_DATA Struct Reference . . . 200
5.69.1 Detailed Description . . . 201
5.69.2 Member Data Documentation . . . 201
5.70 TotalColumnPressure Class Reference . . . 203
5.70.1 Detailed Description . . . 204
5.70.2 Constructor & Destructor Documentation . . . 204
5.70.3 Member Function Documentation . . . 204
5.70.4 Member Data Documentation . . . 204
5.71 TotalPressureIC Class Reference . . . 204
5.71.1 Detailed Description . . . 205
5.71.2 Constructor & Destructor Documentation . . . 205
5.71.3 Member Function Documentation . . . 205
5.71.4 Member Data Documentation . . . 205
5.72 WallAmbientHeatTransfer Class Reference . . . 206
5.72.1 Detailed Description . . . 206
5.72.2 Constructor & Destructor Documentation . . . 207
5.72.3 Member Function Documentation . . . 207

5.72.4 Member Data Documentation 207
5.73 WallHeatAccumulation Class Reference 207
5.73.1 Detailed Description 208
5.73.2 Constructor & Destructor Documentation 208
5.73.3 Member Function Documentation 208
5.73.4 Member Data Documentation 209

6 File Documentation 209
6.1 AdsorbentProperties.h File Reference 209
6.1.1 Detailed Description 209
6.1.2 Function Documentation 210
6.2 AdsorptionHeatAccumulation.h File Reference 210
6.2.1 Detailed Description 211
6.2.2 Function Documentation 211
6.3 AdsorptionMassTransfer.h File Reference 211
6.3.1 Detailed Description 212
6.3.2 Function Documentation 212
6.4 Aux_LDF.h File Reference 212
6.4.1 Detailed Description 212
6.4.2 Function Documentation 213
6.5 BedHeatAccumulation.h File Reference 213
6.5.1 Detailed Description 213
6.5.2 Function Documentation 214
6.6 BedMassAccumulation.h File Reference 214
6.6.1 Detailed Description 214
6.6.2 Function Documentation 215
6.7 BedProperties.h File Reference 215
6.7.1 Function Documentation 215
6.8 BedWallHeatTransfer.h File Reference 215
6.8.1 Detailed Description 215
6.8.2 Function Documentation 216
6.9 ColumnTemperatureIC.h File Reference 216
6.9.1 Detailed Description 216
6.9.2 Function Documentation 217
6.10 ConcentrationIC.h File Reference 217
6.10.1 Detailed Description 217
6.10.2 Function Documentation 218
6.11 CoupledLDF.h File Reference 218
6.11.1 Detailed Description 218
6.11.2 Macro Definition Documentation 219

6.11.3 Function Documentation 219
6.12 DGAdvection.h File Reference 219
6.12.1 Detailed Description 219
6.12.2 Function Documentation 220
6.13 DGAnisotropicDiffusion.h File Reference 220
6.13.1 Detailed Description 220
6.13.2 Function Documentation 221
6.14 DGColumnHeatAdvection.h File Reference 221
6.14.1 Detailed Description 221
6.14.2 Macro Definition Documentation 222
6.14.3 Function Documentation 222
6.15 DGColumnHeatDispersion.h File Reference 222
6.15.1 Detailed Description 223
6.15.2 Macro Definition Documentation 223
6.15.3 Function Documentation 223
6.16 DGColumnMassAdvection.h File Reference 223
6.16.1 Detailed Description 224
6.16.2 Function Documentation 224
6.17 DGColumnMassDispersion.h File Reference 224
6.17.1 Detailed Description 225
6.17.2 Function Documentation 225
6.18 DGColumnWallHeatFluxBC.h File Reference 225
6.18.1 Detailed Description 226
6.18.2 Function Documentation 226
6.19 DGColumnWallHeatFluxLimitedBC.h File Reference 226
6.19.1 Detailed Description 227
6.19.2 Function Documentation 227
6.20 DGFluxBC.h File Reference 227
6.20.1 Detailed Description 228
6.20.2 Function Documentation 228
6.21 DGFluxLimitedBC.h File Reference 228
6.21.1 Detailed Description 229
6.21.2 Function Documentation 229
6.22 DGHeatFluxBC.h File Reference 229
6.22.1 Detailed Description 229
6.22.2 Function Documentation 230
6.23 DGHeatFluxLimitedBC.h File Reference 230
6.23.1 Detailed Description 230
6.23.2 Function Documentation 231
6.24 DGMassFluxBC.h File Reference 231

6.24.1 Detailed Description . . . 231
6.24.2 Function Documentation . . . 232
6.25 DGMassFluxLimitedBC.h File Reference . . . 232
6.25.1 Detailed Description . . . 232
6.25.2 Function Documentation . . . 233
6.26 DgospreyApp.h File Reference . . . 233
6.26.1 Detailed Description . . . 233
6.26.2 Function Documentation . . . 233
6.27 DgospreyRevision.h File Reference . . . 233
6.27.1 Macro Definition Documentation . . . 234
6.28 egret.h File Reference . . . 234
6.28.1 Detailed Description . . . 235
6.28.2 Macro Definition Documentation . . . 235
6.28.3 Function Documentation . . . 237
6.29 error.h File Reference . . . 237
6.29.1 Detailed Description . . . 238
6.29.2 Macro Definition Documentation . . . 239
6.29.3 Enumeration Type Documentation . . . 239
6.29.4 Function Documentation . . . 240
6.30 finch.h File Reference . . . 240
6.30.1 Detailed Description . . . 242
6.30.2 Enumeration Type Documentation . . . 243
6.30.3 Function Documentation . . . 244
6.31 flock.h File Reference . . . 247
6.31.1 Detailed Description . . . 247
6.32 FlowProperties.h File Reference . . . 248
6.32.1 Detailed Description . . . 248
6.32.2 Macro Definition Documentation . . . 249
6.32.3 Function Documentation . . . 249
6.33 GAdvection.h File Reference . . . 249
6.33.1 Detailed Description . . . 249
6.33.2 Function Documentation . . . 250
6.34 GAnisotropicDiffusion.h File Reference . . . 250
6.34.1 Detailed Description . . . 250
6.34.2 Function Documentation . . . 250
6.35 GColumnHeatAdvection.h File Reference . . . 250
6.35.1 Detailed Description . . . 251
6.35.2 Function Documentation . . . 251
6.36 GColumnHeatDispersion.h File Reference . . . 251
6.36.1 Detailed Description . . . 252

6.36.2 Function Documentation 252
6.37 GColumnMassAdvection.h File Reference 252
6.37.1 Detailed Description 252
6.37.2 Function Documentation 253
6.38 GColumnMassDispersion.h File Reference 253
6.38.1 Detailed Description 253
6.38.2 Function Documentation 254
6.39 lark.h File Reference 254
6.39.1 Detailed Description 256
6.39.2 Macro Definition Documentation 258
6.39.3 Enumeration Type Documentation 258
6.39.4 Function Documentation 258
6.40 LinearDrivingForce.h File Reference 269
6.40.1 Detailed Description 269
6.40.2 Function Documentation 269
6.41 macaw.h File Reference 269
6.41.1 Detailed Description 270
6.41.2 Macro Definition Documentation 271
6.42 magpie.h File Reference 271
6.42.1 Detailed Description 273
6.42.2 Macro Definition Documentation 273
6.42.3 Function Documentation 274
6.43 MAGPIE_Adsorption.h File Reference 278
6.43.1 Detailed Description 279
6.43.2 Function Documentation 279
6.44 MAGPIE_AdsorptionHeat.h File Reference 279
6.44.1 Detailed Description 280
6.44.2 Function Documentation 280
6.45 MAGPIE_ConstLDF_Adsorption.h File Reference 280
6.45.1 Detailed Description 281
6.45.2 Macro Definition Documentation 281
6.45.3 Function Documentation 281
6.46 MAGPIE_ConstLDF_Perturbation.h File Reference 281
6.46.1 Detailed Description 282
6.46.2 Function Documentation 282
6.47 MAGPIE_MaterialLDF_Adsorption.h File Reference 282
6.47.1 Detailed Description 283
6.47.2 Function Documentation 283
6.48 MAGPIE_MaterialLDF_Perturbation.h File Reference 283
6.48.1 Detailed Description 284

6.48.2 Function Documentation 284
6.49 MAGPIE_Perturbation.h File Reference 284
6.49.1 Detailed Description 285
6.49.2 Function Documentation 286
6.50 MagpieAdsorbateProperties.h File Reference 286
6.50.1 Detailed Description 286
6.50.2 Function Documentation 286
6.51 scopsowl.h File Reference 286
6.51.1 Detailed Description 288
6.51.2 Macro Definition Documentation 288
6.51.3 Function Documentation 289
6.52 skua.h File Reference 293
6.52.1 Detailed Description 294
6.52.2 Macro Definition Documentation 295
6.52.3 Function Documentation 295
6.53 TotalColumnPressure.h File Reference 299
6.53.1 Detailed Description 299
6.53.2 Function Documentation 299
6.54 TotalPressureIC.h File Reference 299
6.54.1 Detailed Description 300
6.54.2 Function Documentation 300
6.55 WallAmbientHeatTransfer.h File Reference 300
6.55.1 Detailed Description 301
6.55.2 Function Documentation 301
6.56 WallHeatAccumulation.h File Reference 301
6.56.1 Detailed Description 302
6.56.2 Function Documentation 302

Index **302**

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 18
AuxKernel
- **Aux_LDF** 20
  - **MAGPIE_ConstLDF_Adsorption** 142
  - **MAGPIE_ConstLDF_Perturbation** 144
  - **MAGPIE_MaterialLDF_Adsorption** 147
  - **MAGPIE_MaterialLDF_Perturbation** 149
- **MAGPIE_Adsorption** 139
- **MAGPIE_AdsorptionHeat** 141
- **MAGPIE_Perturbation** 152
- **TotalColumnPressure** 203

BACKTRACK_DATA 21
BiCGSTAB_DATA 31
CGS_DATA 35
DGKernel
- **DGAdvection** 43
  - **DGColumnHeatAdvection** 48
  - **DGColumnMassAdvection** 53
- **DGAnisotropicDiffusion** 45
  - **DGColumnHeatDispersion** 50
  - **DGColumnMassDispersion** 55

FINCH_DATA 86

GCR_DATA 119
GMRESLP_DATA 123
GMRESR_DATA 125
GMRESRP_DATA 128
GPAST_DATA 132
GSTA_DATA 134
InitialCondition
- **ColumnTemperatureIC** 38
- **ConcentrationIC** 40
- **TotalPressureIC** 204

IntegratedBC
- **DGFluxBC** 65
  - **DGColumnWallHeatFluxBC** 58
  - **DGHeatFluxBC** 70
  - **DGMassFluxBC** 77
- **DGFluxLimitedBC** 67
  - **DGColumnWallHeatFluxLimitedBC** 61
  - **DGHeatFluxLimitedBC** 74
  - **DGMassFluxLimitedBC** 81

Kernel
- **AdsorptionHeatAccumulation** 15
- **AdsorptionMassTransfer** 16
- **BedWallHeatTransfer** 29
- **GAdvection** 106
  - **GColumnHeatAdvection** 110
  - **GColumnMassAdvection** 115
- **GAnisotropicDiffusion** 108
  - **GColumnHeatDispersion** 112
  - **GColumnMassDispersion** 117
- **LinearDrivingForce** 138
  - **CoupledLDF** 41
- **WallAmbientHeatTransfer** 206

KMS_DATA 135

MAGPIE_DATA 146
Material
  AdsorbentProperties 10
  BedProperties 26
  FlowProperties 100
  MagpieAdsorbateProperties 154
Matrix< T > 158
Matrix< double > 158
MIXED_GAS 169
MooseApp
  DgospreyApp 85
mSPD_DATA 172
NUM_JAC_DATA 173
OPTRANS_DATA 174
PCG_DATA 174
PICARD_DATA 177
PJFNK_DATA 179
PURE_GAS 184
SCOPSOWL_DATA 186
SCOPSOWL_PARAM_DATA 192
SKUA_DATA 195
SKUA_PARAM 199
SYSTEM_DATA 200
TimeDerivative
  BedHeatAccumulation 23
  BedMassAccumulation 24
  WallHeatAccumulation 207

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 15

AdsorptionMassTransfer
AdsorptionMassTransfer class object inherits from Kernel object 16

ARNOLDI_DATA
Data structure for the construction of the Krylov subspaces for a linear system 18

Aux_LDF
Aux_LDF class inherits from AuxKernel 20

BACKTRACK_DATA
Data structure for the implementation of Backtracking Linesearch 21

BedHeatAccumulation
BedHeatAccumulation class object inherits from TimeDerivative object 23

BedMassAccumulation
BedMassAccumulation class object inherits from TimeDerivative object 24

BedProperties
BedProperties class object inherits from Material object 26

BedWallHeatTransfer
BedWallHeatTransfer class object inherits from Kernel object 29

BiCGSTAB_DATA
Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems 31

CGS_DATA
Data structure for the implementation of the CGS algorithm for non-symmetric linear systems 35

ColumnTemperatureIC
ColumnTemperatureIC class object inherits from InitialCondition object 38

ConcentrationIC
ConcentrationIC class object inherits from InitialCondition object 40

CoupledLDF
CoupledLDF class object inherits from LinearDrivingForce object 41

DGAdvection
DGAdvection class object inherits from DGKernel object 43

DGAnisotropicDiffusion
DGAnisotropicDiffusion class object inherits from DGKernel object 45

DGColumnHeatAdvection
DGColumnHeatAdvection class object inherits from DGAdvection object 48

DGColumnHeatDispersion
DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object 50

DGColumnMassAdvection
DGColumnMassAdvection class object inherits from DGAdvection object 53

DGColumnMassDispersion
DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object 55

DGColumnWallHeatFluxBC
DGColumnWallHeatFluxBC class object inherits from DGFluxBC object 58

DGColumnWallHeatFluxLimitedBC
DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object 61

DGFluxBC
DGFluxBC class object inherits from IntegratedBC object 65

DGFluxLimitedBC
DGFluxLimitedBC class object inherits from IntegratedBC object 67

DGHeatFluxBC
DGHeatFluxBC class object inherits from DGFluxBC object 70

DGHeatFluxLimitedBC
DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object 74

DGMassFluxBC
DGMassFluxBC class object inherits from DGFluxBC object 77

DGMassFluxLimitedBC
DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object 81

DgospreyApp
DgospreyApp inherits from the MooseApp object 85

FINCH_DATA
Data structure for the FINCH object 86

FlowProperties
FlowProperties class object inherits from Material object 100

GAdvection
GAdvection class object inherits from Kernel object 106

GAnisotropicDiffusion
GAnisotropicDiffusion class object inherits from Kernel object 108

GColumnHeatAdvection
GColumnHeatAdvection class object inherits from GAdvection object 110

GColumnHeatDispersion
GColumnHeatDispersion class object inherits from GAnisotropicDiffusion object 112

GColumnMassAdvection
GColumnMassAdvection class object inherits from GAdvection object 115

GColumnMassDispersion
GColumnMassDispersion class object inherits from GAnisotropicDiffusion object 117

GCR_DATA
Data structure for the implementation of the GCR algorithm for non-symmetric linear systems 119

GMRESLP_DATA
Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning 123

GMRESR_DATA
Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMR-ESR) 125

GMRESRP_DATA
Data structure for the Restarted GMRES algorithm with Right Preconditioning 128
GPAST_DATA
GPAST Data Structure 132
GSTA_DATA
GSTA Data Structure 134
KMS_DATA
Data structure for the implemenation of the Krylov Multi-Space (KMS) Method 135
LinearDrivingForce
LinearDrivingForce class object inherits from Kernel object 138
MAGPIE_Adsorption
Magpie Adsorption class inherits from AuxKernel 139
MAGPIE_AdsorptionHeat
Magpie Adsorption Heat class inherits from AuxKernel 141
MAGPIE_ConstLDF_Adsorption
MAGPIE_ConstLDF class inherits from AuxKernel 142
MAGPIE_ConstLDF_Perturbation
MAGPIE_ConstLDF class inherits from AuxKernel 144
MAGPIE_DATA
MAGPIE Data Structure 146
MAGPIE_MaterialLDF_Adsorption
MAGPIE_MaterialLDF_Adsorption class inherits from AuxKernel 147
MAGPIE_MaterialLDF_Perturbation
MAGPIE_MaterialLDF_Perturbation class inherits from AuxKernel 149
MAGPIE_Perturbation
Magpie Perturbation class inherits from AuxKernel 152
MagpieAdsorbateProperties
MagpieAdsorbateProperties class object inherits from Material object 154
Matrix$< T >$
Templated C++ Matrix Class Object (click Matrix to go to function definitions) 158
MIXED_GAS
Data structure holding information necessary for computing mixed gas properties 169
mSPD_DATA
MSPD Data Structure 172
NUM_JAC_DATA
Data structure to form a numerical jacobian matrix with finite differences 173
OPTRANS_DATA
Data structure for implementation of linear operator transposition 174
PCG_DATA
Data structure for implementation of the PCG algorithms for symmetric linear systems 174
PICARD_DATA
Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems 177

PJFNK_DATA
Data structure for the implementation of the PJFNK algorithm for non-linear systems 179
PURE_GAS
Data structure holding all the parameters for each pure gas spieces 184
SCOPSOWL_DATA
Primary data structure for SCOPSOWL simulations 186
SCOPSOWL_PARAM_DATA
Data structure for the species' parameters in SCOPSOWL 192
SKUA_DATA
Data structure for all simulation information in SKUA 195
SKUA_PARAM
Data structure for species' parameters in SKUA 199
SYSTEM_DATA
System Data Structure 200
TotalColumnPressure
Total Column Pressure class inherits from AuxKernel 203
TotalPressureIC
TotalPressureIC class object inherits from InitialCondition object 204
WallAmbientHeatTransfer
WallAmbientHeatTransfer class object inherits from Kernel object 206
WallHeatAccumulation
WallHeatAccumulation class object inherits from TimeDerivative object 207

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 209
AdsorptionHeatAccumulation.h
Standard kernel for the heat of adsorption and its effect on the system temperature 210
AdsorptionMassTransfer.h
Standard kernel for the transfer of mass via adsorption 211
Aux_LDF.h
Generic auxillary kernel to calculate the value of an aux variable using LDF kinetics 212
BedHeatAccumulation.h
Time Derivative kernel for the accumulation of heat in a fixed-bed column 213
BedMassAccumulation.h
Time Derivative kernel for the accumulation of mass of a species in a fixed-bed column 214
BedProperties.h 215

BedWallHeatTransfer.h
Standard kernel for the transfer of heat from the fixed-bed to the column wall 215

ColumnTemperatureIC.h
Initial Condition kernel for initial temperature in a fixed-bed column 216

ConcentrationIC.h
Initial Condition kernel for initial concentration of a species in a fixed-bed column 217

CoupledLDF.h
Advanced kernel for a cross coupled linear driving force mechanism 218

DGAdvection.h
Discontinous Galerkin kernel for advection 219

DGAnisotropicDiffusion.h
Discontinous Galerkin kernel for anisotropic diffusion 220

DGColumnHeatAdvection.h
Discontinous Galerkin kernel for energy advection in a fixed-bed column 221

DGColumnHeatDispersion.h
Discontinous Galerkin kernel for energy dispersion in a fixed-bed column 222

DGColumnMassAdvection.h
Discontinous Galerkin kernel for mass advection in a fixed-bed column 223

DGColumnMassDispersion.h
Discontinous Galerkin kernel for mass dispersion in a fixed-bed column 224

DGColumnWallHeatFluxBC.h
Boundary Condition kernel for the heat flux across the wall of the fixed-bed column 225

DGColumnWallHeatFluxLimitedBC.h
Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall 226

DGFluxBC.h
Boundary Condition kernel for the flux across a boundary of the domain 227

DGFluxLimitedBC.h
Boundary Condition kernel to mimic a Dirichlet BC for DG methods 228

DGHeatFluxBC.h
Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column 229

DGHeatFluxLimitedBC.h
Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet 230

DGMassFluxBC.h
Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column 231

DGMassFluxLimitedBC.h
Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet 232

DgospreyApp.h
Registration object for creating a registering DGOSPREY kernels 233

DgospreyRevision.h 233

egret.h
Estimation of Gas-phase pRopErTies 234

error.h
All error types are defined here 237

finch.h
Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme 240

flock.h
FundamentaL Off-gas Collection of Kernels 247

FlowProperties.h
Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics 248

GAdvection.h
Kernel for use with the corresponding DGAdvection object 249

GAnisotropicDiffusion.h
Kernel for use with the corresponding DGAnisotropicDiffusion object 250

GColumnHeatAdvection.h
Kernel for use with the corresponding DGColumnHeatAdvection object 250

GColumnHeatDispersion.h
Kernel for use with the corresponding DGColumnHeatDispersion object 251

GColumnMassAdvection.h
Kernel for use with the corresponding DGColumnMassAdvection object 252

GColumnMassDispersion.h
Kernel for use with the corresponding DGColumnMassDispersion object 253

lark.h
Linear Algebra Residual Kernels 254

LinearDrivingForce.h
Standard kernel for a generic coupled linear driving force mechanism 269

macaw.h
MAtrix CAlculation Workspace 269

magpie.h
Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria 271

MAGPIE_Adsorption.h
Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system 278

MAGPIE_AdsorptionHeat.h
Auxillary kernel to calculate heat of adsorption of a particular gas species in the system 279

MAGPIE_ConstLDF_Adsorption.h
Auxillary kernel to calculate adsorption based on LDF kinetics with constant coefficients 280

MAGPIE_ConstLDF_Perturbation.h
Auxillary kernel to calculate adsorption perturbation based on LDF kinetics with constant coefficients 281

MAGPIE_MaterialLDF_Adsorption.h
Auxillary kernel to calculate adsorption based on LDF kinetics with material property coefficients 282

MAGPIE_MaterialLDF_Perturbation.h
Auxillary kernel to calculate adsorption perturbation based on LDF kinetics with material property coefficients 283

MAGPIE_Perturbation.h
Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system 284

MagpieAdsorbateProperties.h
Material Properties kernel that will setup and hold all information associated with MAGPIE simulations 286

scopsowl.h
Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems 286

skua.h
Surface Kinetics for Uptake by Adsorption 293

TotalColumnPressure.h
Auxillary kernel to calculate total column pressure based on temperature and concentrations 299

TotalPressureIC.h
Initial Condition kernel for initial temperature in a fixed-bed column 299

WallAmbientHeatTransfer.h
Standard kernel for the transfer of heat from the column wall to the ambient air 300

WallHeatAccumulation.h
Time Derivative kernel for the accumulation of heat in a walls of the column 301

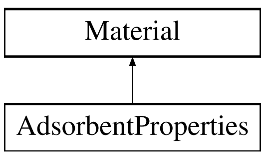
5 Class Documentation

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 _eps_binder

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

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

  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)
- std::vector< Real > _ref_diff

  Reference Surface Diffusivity (um^2/hr)
- std::vector< Real > _act_energy

  Activation Energy of Surface Diffusion (J/mol)
- std::vector< Real > _ref_temp

  Reference Temperature for Surface Diffusion (K)
- std::vector< Real > _affinity

  Affinity coefficient for Surface Diffusion (-)
- 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)
- MaterialProperty< Real > & _pellet_diameter

  MaterialProperty for pellet diameter (cm)
- MaterialProperty< Real > & _crystal_radius

  MaterialProperty for the crystal radius (um)
- MaterialProperty< Real > & _binder_porosity

  MaterialProperty for the binder porosity.
- MaterialProperty< Real > & _binder_ratio

  MaterialProperty for the ratio of binder to pellet volumes.
- MaterialProperty< Real > & _pore_size

  MaterialProperty for the macropore radius (cm)
- MaterialProperty< std::vector
  < Real > > & _surface_diffusion

  MaterialProperty for the surface diffusion (um^2/hr)
- const MaterialProperty
  < MAGPIE_DATA > & _magpie_dat

  Pointer to MAGPIE_DATA material property.
- 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 58 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 std::vector<Real> AdsorbentProperties::_act_energy [private]

Activation Energy of Surface Diffusion (J/mol)

Definition at line 79 of file AdsorbentProperties.h.

5.1.4.2 std::vector<Real> AdsorbentProperties::_affinity [private]

Affinity coefficient for Surface Diffusion (-)

Definition at line 81 of file AdsorbentProperties.h.

5.1.4.3 Real AdsorbentProperties::_binder_fraction [private]

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

Definition at line 70 of file AdsorbentProperties.h.

5.1.4.4 MaterialProperty<Real>& AdsorbentProperties::_binder_porosity [private]

MaterialProperty for the binder porosity.

Definition at line 87 of file AdsorbentProperties.h.

5.1.4.5 MaterialProperty<Real>& AdsorbentProperties::_binder_ratio [private]

MaterialProperty for the ratio of binder to pellet volumes.

Definition at line 88 of file AdsorbentProperties.h.

5.1.4.6 Real AdsorbentProperties::_crystal_rad [private]

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

Definition at line 72 of file AdsorbentProperties.h.

5.1.4.7 MaterialProperty<Real>& AdsorbentProperties::_crystal_radius [private]

MaterialProperty for the crystal radius (um)

Definition at line 86 of file AdsorbentProperties.h.

5.1.4.8 Real AdsorbentProperties::_eps_binder [private]

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

Definition at line 71 of file AdsorbentProperties.h.

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

Pointer list for the coupled gases.

Definition at line 97 of file AdsorbentProperties.h.

5.1.4.10 Real AdsorbentProperties::_hs [private]

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

Definition at line 76 of file AdsorbentProperties.h.

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

List of indices for the coupled gases.

Definition at line 96 of file AdsorbentProperties.h.

5.1.4.12 Real AdsorbentProperties::_macropore_radius [private]

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

Definition at line 74 of file AdsorbentProperties.h.

5.1.4.13 const MaterialProperty< MAGPIE_DATA >& AdsorbentProperties::_magpie_dat [private]

Pointer to MAGPIE_DATA material property.

Definition at line 93 of file AdsorbentProperties.h.

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

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

Definition at line 83 of file AdsorbentProperties.h.

5.1.4.15 Real AdsorbentProperties::_pellet_dia [private]

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

Definition at line 73 of file AdsorbentProperties.h.

5.1.4.16 MaterialProperty<Real>& AdsorbentProperties::_pellet_diameter [private]

MaterialProperty for pellet diameter (cm)

Definition at line 85 of file AdsorbentProperties.h.

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

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

Definition at line 84 of file AdsorbentProperties.h.

5.1.4.18 MaterialProperty<Real>& AdsorbentProperties::_pore_size [private]

MaterialProperty for the macropore radius (cm)

Definition at line 89 of file AdsorbentProperties.h.

5.1.4.19 std::vector<Real> AdsorbentProperties::_ref_diff [private]

Reference Surface Diffusivity (um^2/hr)

Definition at line 78 of file AdsorbentProperties.h.

5.1.4.20 std::vector<Real> AdsorbentProperties::_ref_temp [private]

Reference Temperature for Surface Diffusion (K)

Definition at line 80 of file AdsorbentProperties.h.

5.1.4.21 Real AdsorbentProperties::_rhos [private]

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

Definition at line 75 of file AdsorbentProperties.h.

5.1.4.22 MaterialProperty<std::vector<Real> >& AdsorbentProperties::_surface_diffusion [private]

MaterialProperty for the surface diffusion (um^2/hr)

Definition at line 91 of file AdsorbentProperties.h.

5.1.4.23 VariableValue& AdsorbentProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 95 of file AdsorbentProperties.h.

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

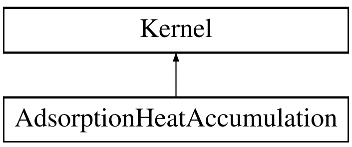
- AdsorbentProperties.h

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:

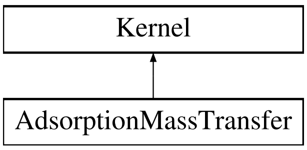
- AdsorptionHeatAccumulation.h

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:

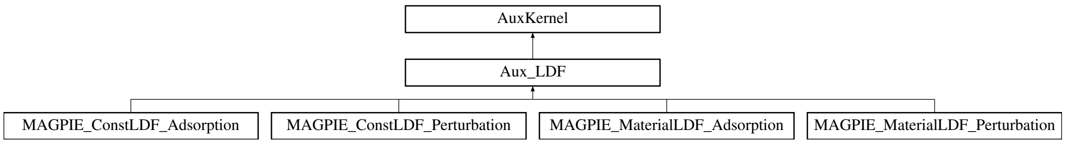
- lark.h

5.5 Aux_LDF Class Reference

Aux_LDF class inherits from AuxKernel.

`#include <Aux_LDF.h>`

Inheritance diagram for Aux_LDF:



Public Member Functions

- Aux_LDF (const InputParameters &parameters)

  Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

  Required MOOSE function override.

Protected Attributes

- Real _ldf_coef
- Real _driving_value

  Value of the driving force coefficient.

5.5.1 Detailed Description

Aux_LDF class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the result of the linear driving force function, integrated implicitly, for the aux variable it is associated with. It contains two parameters: (i) the ldf coefficient and (ii) the driving value. Inherit from this base class to alter the parameters and change the behavior of this kernel to fit your particular problem.

Definition at line 58 of file Aux_LDF.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Aux_LDF::Aux_LDF (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.5.3 Member Function Documentation

5.5.3.1 virtual Real Aux_LDF::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.

Reimplemented in MAGPIE_MaterialLDF_Adsorption, MAGPIE_MaterialLDF_Perturbation, MAGPIE_ConstLDF_-Adsorption, and MAGPIE_ConstLDF_Perturbation.

5.5.4 Member Data Documentation

5.5.4.1 Real Aux_LDF::_driving_value [protected]

Value of the driving force coefficient.

Definition at line 71 of file Aux_LDF.h.

5.5.4.2 Real Aux_LDF::_ldf_coef [protected]

Definition at line 70 of file Aux_LDF.h.

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

- Aux_LDF.h

5.6 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.

5.6.1 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.

5.6.2 Member Data Documentation

5.6.2.1 double BACKTRACK_DATA::alpha = 1e-4

Scaling parameter for determination of search step size.

Definition at line 477 of file lark.h.

5.6.2.2 bool BACKTRACK_DATA::constRho = false

True = use a constant value for rho.

Definition at line 482 of file lark.h.

5.6.2.3 Matrix<double> BACKTRACK_DATA::Fk

Old residual vector of the Newton step.

Definition at line 484 of file lark.h.

5.6.2.4 int BACKTRACK_DATA::fun_call = 0

Number of function calls made during line search.

Definition at line 476 of file lark.h.

5.6.2.5 double BACKTRACK_DATA::lambdaMin =DBL_EPSILON

Smallest allowable step length.

Definition at line 479 of file lark.h.

5.6.2.6 double BACKTRACK_DATA::normFkp1

New residual norm of the Newton step.

Definition at line 480 of file lark.h.

5.6.2.7 double BACKTRACK_DATA::rho = 0.1

Scaling parameter for to change step size by.

Definition at line 478 of file lark.h.

5.6.2.8 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:

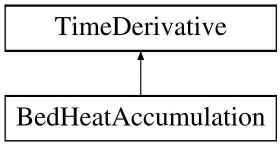
- lark.h

5.7 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.

5.7.1 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.

5.7.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.7.3 Member Function Documentation

5.7.3.1 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.

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

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.7.4 Member Data Documentation

5.7.4.1 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:

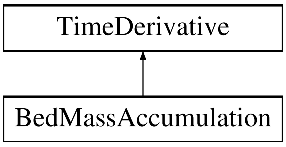
- BedHeatAccumulation.h

5.8 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.

5.8.1 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.

5.8.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.8.3 Member Function Documentation

5.8.3.1 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.

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

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.8.4 Member Data Documentation

5.8.4.1 int BedMassAccumulation::_index [private]

Index of the species of interest for the mass accumulation.

Definition at line 71 of file BedMassAccumulation.h.

5.8.4.2 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:

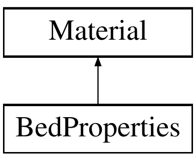
- BedMassAccumulation.h

5.9 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.

5.9.1 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.

5.9.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.9.3 Member Function Documentation

5.9.3.1 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.

5.9.4 Member Data Documentation

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

MaterialProperty for bed-wall heat transfer coefficient.

Definition at line 83 of file BedProperties.h.

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

MaterialProperty for thermal conductivity of the bed.

Definition at line 80 of file BedProperties.h.

5.9.4.3 Real BedProperties::_din [private]

Column inner diameter (cm)

Definition at line 68 of file BedProperties.h.

5.9.4.4 Real BedProperties::_dout [private]

Column outer diameter (cm)

Definition at line 69 of file BedProperties.h.

5.9.4.5 Real BedProperties::_eb [private]

Bulk porosity of the bed.

Definition at line 70 of file BedProperties.h.

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

Pointer list of the gas species concentrations.

Definition at line 88 of file BedProperties.h.

5.9.4.7 Real BedProperties::_hw [private]

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

Definition at line 73 of file BedProperties.h.

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

List of indices for the species in the system.

Definition at line 87 of file BedProperties.h.

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

MaterialProperty for column inner diameter.

Definition at line 77 of file BedProperties.h.

5.9.4.10 Real BedProperties::_Kz [private]

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

Definition at line 71 of file BedProperties.h.

5.9.4.11 Real BedProperties::_length [private]

Bed length (cm)

Definition at line 67 of file BedProperties.h.

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

MaterialProperty for column outer diameter.

Definition at line 78 of file BedProperties.h.

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

MaterialProperty for bulk porosity of the bed.

Definition at line 79 of file BedProperties.h.

5.9.4.14 Real BedProperties::_rhow [private]

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

Definition at line 72 of file BedProperties.h.

5.9.4.15 VariableValue& BedProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 86 of file BedProperties.h.

5.9.4.16 Real BedProperties::_Ua [private]

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

Definition at line 75 of file BedProperties.h.

5.9.4.17 Real BedProperties::_Uw [private]

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

Definition at line 74 of file BedProperties.h.

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

MaterialProperty for column wall density.

Definition at line 81 of file BedProperties.h.

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

MaterialProperty for exterior-wall heat transfer coefficient.

Definition at line 84 of file BedProperties.h.

5.9.4.20 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:

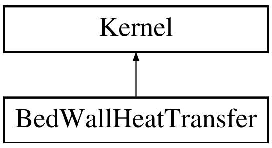
- BedProperties.h

5.10 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.

5.10.1 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.

5.10.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.10.3 Member Function Documentation

5.10.3.1 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.

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

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.10.4 Member Data Documentation

5.10.4.1 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.

5.10.4.2 VariableValue& BedWallHeatTransfer::_column_temp [private]

Reference to the gas temperature coupled non-linear variable.

Definition at line 77 of file BedWallHeatTransfer.h.

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

Reference to the wall inner diameter material property.

Definition at line 74 of file BedWallHeatTransfer.h.

5.10.4.4 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

5.11 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.

5.11.1 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.

5.11.2 Member Data Documentation

5.11.2.1 double BiCGSTAB_DATA::alpha

Step size parameter for next solution.

Definition at line 255 of file lark.h.

5.11.2.2 double BiCGSTAB_DATA::bestres

Best found residual norm.

Definition at line 266 of file lark.h.

5.11.2.3 Matrix<double> BiCGSTAB_DATA::bestx

Best found solution to the linear system.

Definition at line 271 of file lark.h.

5.11.2.4 double BiCGSTAB_DATA::beta

Step size parameter for search direction.

Definition at line 256 of file lark.h.

5.11.2.5 bool BiCGSTAB_DATA::breakdown

Boolean to determine if the method broke down.

Definition at line 253 of file lark.h.

5.11.2.6 int BiCGSTAB_DATA::iter = 0

Actual number of iterations.

Definition at line 252 of file lark.h.

5.11.2.7 int BiCGSTAB_DATA::maxit = 0

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

Definition at line 251 of file lark.h.

5.11.2.8 double BiCGSTAB_DATA::omega

Scaling parameter and additional step length.

Definition at line 259 of file lark.h.

5.11.2.9 double BiCGSTAB_DATA::omega_old

Previous scaling parameter and step length.

Definition at line 260 of file lark.h.

5.11.2.10 bool BiCGSTAB_DATA::Output = true

True = print messages to console.

Definition at line 268 of file lark.h.

5.11.2.11 Matrix<double> BiCGSTAB_DATA::p

Search direction for updating.

Definition at line 275 of file lark.h.

5.11.2.12 Matrix<double> BiCGSTAB_DATA::r

Residual vector for the linear system.

Definition at line 272 of file lark.h.

5.11.2.13 Matrix<double> BiCGSTAB_DATA::r0

Initial residual vector.

Definition at line 273 of file lark.h.

5.11.2.14 double BiCGSTAB_DATA::relres

Relative residual norm.

Definition at line 264 of file lark.h.

5.11.2.15 double BiCGSTAB_DATA::relres_base

Initial residual norm.

Definition at line 265 of file lark.h.

5.11.2.16 double BiCGSTAB_DATA::res

Absolute residual norm.

Definition at line 263 of file lark.h.

5.11.2.17 double BiCGSTAB_DATA::rho

Scaling parameter for alpha and beta.

Definition at line 257 of file lark.h.

5.11.2.18 double BiCGSTAB_DATA::rho_old

Previous scaling parameter for alpha and beta.

Definition at line 258 of file lark.h.

5.11.2.19 Matrix<double> BiCGSTAB_DATA::s

Residual updating vector.

Definition at line 277 of file lark.h.

5.11.2.20 Matrix<double> BiCGSTAB_DATA::t

Search direction for resdidual updates.

Definition at line 279 of file lark.h.

5.11.2.21 double BiCGSTAB_DATA::tol_abs = 1e-6

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

Definition at line 262 of file lark.h.

5.11.2.22 double BiCGSTAB_DATA::tol_rel = 1e-6

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

Definition at line 261 of file lark.h.

5.11.2.23 Matrix<double> BiCGSTAB_DATA::v

Search direction for p.

Definition at line 274 of file lark.h.

5.11.2.24 Matrix<double> BiCGSTAB_DATA::x

Current solution to the linear system.

Definition at line 270 of file lark.h.

5.11.2.25 Matrix<double> BiCGSTAB_DATA::y

Preconditioned search direction.

Definition at line 276 of file lark.h.

5.11.2.26 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

5.12 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.

5.12.1 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.

5.12.2 Member Data Documentation

5.12.2.1 double CGS_DATA::alpha

Step size parameter for next solution.

Definition at line 294 of file lark.h.

5.12.2.2 double CGS_DATA::bestres

Best found residual norm.

Definition at line 303 of file lark.h.

5.12.2.3 Matrix<double> CGS_DATA::bestx

Best found solution to the linear system.

Definition at line 308 of file lark.h.

5.12.2.4 double CGS_DATA::beta

Step size parameter for search direction.

Definition at line 295 of file lark.h.

5.12.2.5 bool CGS_DATA::breakdown

Boolean to determine if the method broke down.

Definition at line 292 of file lark.h.

5.12.2.6 Matrix<double> CGS_DATA::c

Holds the matvec result between A and p.

Definition at line 315 of file lark.h.

5.12.2.7 int CGS_DATA::iter = 0

Actual number of iterations.

Definition at line 291 of file lark.h.

5.12.2.8 int CGS_DATA::maxit = 0

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

Definition at line 290 of file lark.h.

5.12.2.9 bool CGS_DATA::Output = true

True = print messages to console.

Definition at line 305 of file lark.h.

5.12.2.10 Matrix<double> CGS_DATA::p

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

Definition at line 314 of file lark.h.

5.12.2.11 Matrix<double> CGS_DATA::r

Residual vector for the linear system.

Definition at line 309 of file lark.h.

5.12.2.12 Matrix<double> CGS_DATA::r0

Initial residual vector.

Definition at line 310 of file lark.h.

5.12.2.13 double CGS_DATA::relres

Relative residual norm.

Definition at line 301 of file lark.h.

5.12.2.14 double CGS_DATA::relres_base

Initial residual norm.

Definition at line 302 of file lark.h.

5.12.2.15 double CGS_DATA::res

Absolute residual norm.

Definition at line 300 of file lark.h.

5.12.2.16 double CGS_DATA::rho

Scaling parameter for alpha and beta.

Definition at line 296 of file lark.h.

5.12.2.17 double CGS_DATA::sigma

Scaling parameter and additional step length.

Definition at line 297 of file lark.h.

5.12.2.18 double CGS_DATA::tol_abs = 1e-6

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

Definition at line 299 of file lark.h.

5.12.2.19 double CGS_DATA::tol_rel = 1e-6

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

Definition at line 298 of file lark.h.

5.12.2.20 Matrix<double> CGS_DATA::u

Search direction for v.

Definition at line 311 of file lark.h.

5.12.2.21 Matrix<double> CGS_DATA::v

Search direction for x.

Definition at line 313 of file lark.h.

5.12.2.22 Matrix<double> CGS_DATA::w

Updates sigma and u.

Definition at line 312 of file lark.h.

5.12.2.23 Matrix<double> CGS_DATA::x

Current solution to the linear system.

Definition at line 307 of file lark.h.

5.12.2.24 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:

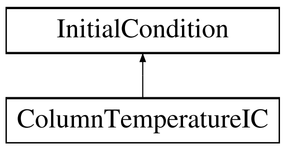
- lark.h

5.13 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)

5.13.1 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.

5.13.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.13.3 Member Function Documentation

5.13.3.1 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.

5.13.4 Member Data Documentation

5.13.4.1 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:

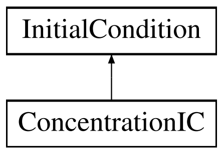
- ColumnTemperatureIC.h

5.14 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)

5.14.1 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.

5.14.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.14.3 Member Function Documentation

5.14.3.1 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.

5.14.4 Member Data Documentation

5.14.4.1 Real ConcentrationIC::_PT_IC [private]

Initial total pressure in the column (kPa)

Definition at line 71 of file ConcentrationIC.h.

5.14.4.2 Real ConcentrationIC::_T_IC [private]

Initial temperature in the column (K)

Definition at line 72 of file ConcentrationIC.h.

5.14.4.3 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:

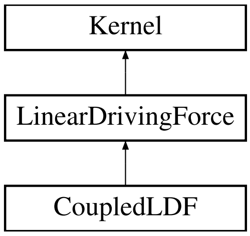
- ConcentrationIC.h

5.15 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.

5.15.1 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.

5.15.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.15.3 Member Function Documentation

5.15.3.1 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.

5.15.3.2 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.

5.15.4 Member Data Documentation

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

Coefficient for the strength or rate of the driving force.

Definition at line 77 of file LinearDrivingForce.h.

5.15.4.2 Real CoupledLDF::_drive_coef [protected]

Coefficient for relationship between coupled variables.

Definition at line 80 of file CoupledLDF.h.

5.15.4.3 VariableValue& CoupledLDF::_drive_var [protected]

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

Definition at line 81 of file CoupledLDF.h.

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

Value the coupled variable is driving towards.

Definition at line 78 of file LinearDrivingForce.h.

5.15.4.5 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.

5.15.4.6 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:

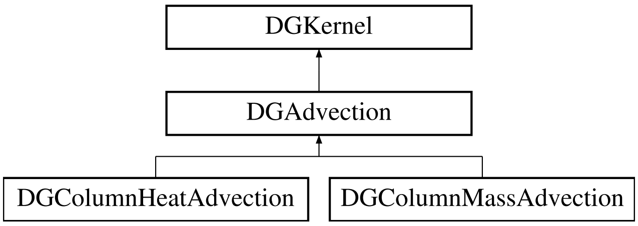
- CoupledLDF.h

5.16 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)

5.16.1 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.

5.16.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.16.3 Member Function Documentation

5.16.3.1 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.

5.16.3.2 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.

5.16.4 Member Data Documentation

5.16.4.1 RealVectorValue DGAdvection::_velocity [protected]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

5.16.4.2 Real DGAdvection::_vx [protected]

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

Definition at line 83 of file DGAdvection.h.

5.16.4.3 Real DGAdvection::_vy [protected]

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

Definition at line 84 of file DGAdvection.h.

5.16.4.4 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:

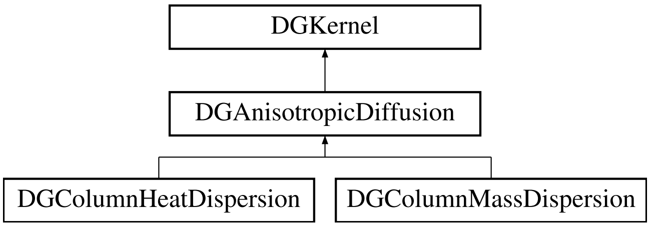
- DGAdvection.h

5.17 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

5.17.1 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.

5.17.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.17.3 Member Function Documentation

5.17.3.1 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.

5.17.3.2 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.

5.17.4 Member Data Documentation

5.17.4.1 RealTensorValue DGAnisotropicDiffusion::_Diffusion [protected]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

5.17.4.2 Real DGAnisotropicDiffusion::_Dxx [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.17.4.3 Real DGAnisotropicDiffusion::_Dxy [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.17.4.4 Real DGAnisotropicDiffusion::_Dxz [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

5.17.4.5 Real DGAnisotropicDiffusion::_Dyx [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.17.4.6 Real DGAnisotropicDiffusion::_Dyy [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.17.4.7 Real DGAnisotropicDiffusion::_Dyz [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

5.17.4.8 Real DGAnisotropicDiffusion::_Dzx [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.17.4.9 Real DGAnisotropicDiffusion::_Dzy [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.17.4.10 Real DGAnisotropicDiffusion::_Dzz [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

5.17.4.11 Real DGAnisotropicDiffusion::_epsilon [protected]

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

Definition at line 82 of file DGAnisotropicDiffusion.h.

5.17.4.12 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:

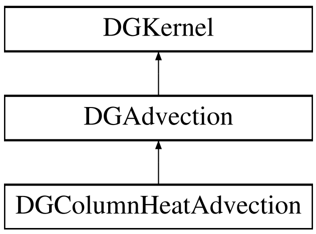
- DGAnisotropicDiffusion.h

5.18 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.

5.18.1 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.

5.18.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.18.3 Member Function Documentation

5.18.3.1 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.

5.18.3.2 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.

5.18.4 Member Data Documentation

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

Reference to the gas density material property.

Definition at line 82 of file DGColumnHeatAdvection.h.

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

Reference to the gas heat capacity material property.

Definition at line 83 of file DGColumnHeatAdvection.h.

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

Reference to the velocity material property.

Definition at line 81 of file DGColumnHeatAdvection.h.

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

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

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

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

Definition at line 83 of file DGAdvection.h.

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

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

Definition at line 84 of file DGAdvection.h.

5.18.4.7 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:

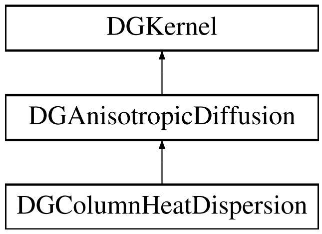
- DGColumnHeatAdvection.h

5.19 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.

5.19.1 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 discontinuous 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.

5.19.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.19.3 Member Function Documentation

5.19.3.1 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.

5.19.3.2 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.

5.19.4 Member Data Documentation

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

Reference to the thermal conductivity material property.

Definition at line 81 of file DGColumnHeatDispersion.h.

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

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

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

Definition at line 86 of file DGAnisotropicDiffusion.h.

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

Definition at line 86 of file DGAnisotropicDiffusion.h.

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

Definition at line 86 of file DGAnisotropicDiffusion.h.

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

Definition at line 87 of file DGAnisotropicDiffusion.h.

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

Definition at line 87 of file DGAnisotropicDiffusion.h.

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

Definition at line 87 of file DGAnisotropicDiffusion.h.

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

Definition at line 88 of file DGAnisotropicDiffusion.h.

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

Definition at line 88 of file DGAnisotropicDiffusion.h.

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

Definition at line 88 of file DGAnisotropicDiffusion.h.

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

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

Definition at line 82 of file DGAnisotropicDiffusion.h.

5.19.4.13 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:

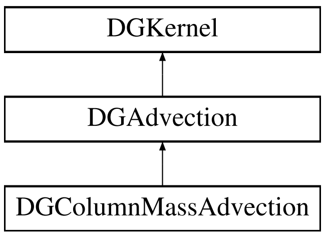
- DGColumnHeatDispersion.h

5.20 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.

5.20.1 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.

5.20.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.20.3 Member Function Documentation

5.20.3.1 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.

5.20.3.2 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.

5.20.4 Member Data Documentation

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

Reference to the velocity material property.

Definition at line 80 of file DGColumnMassAdvection.h.

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

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

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

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

Definition at line 83 of file DGAdvection.h.

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

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

Definition at line 84 of file DGAdvection.h.

5.20.4.5 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:

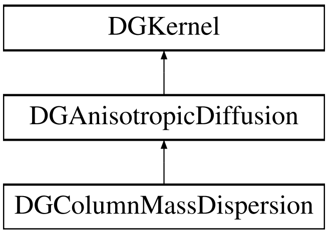
- DGColumnMassAdvection.h

5.21 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.

5.21.1 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.

5.21.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.21.3 Member Function Documentation

5.21.3.1 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.

5.21.3.2 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.

5.21.4 Member Data Documentation

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

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

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

Reference to the axial dispersion material property.

Definition at line 83 of file DGColumnMassDispersion.h.

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

Definition at line 86 of file DGAnisotropicDiffusion.h.

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

Definition at line 86 of file DGAnisotropicDiffusion.h.

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

Definition at line 86 of file DGAnisotropicDiffusion.h.

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

Definition at line 87 of file DGAnisotropicDiffusion.h.

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

Definition at line 87 of file DGAnisotropicDiffusion.h.

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

Definition at line 87 of file DGAnisotropicDiffusion.h.

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

Definition at line 88 of file DGAnisotropicDiffusion.h.

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

Definition at line 88 of file DGAnisotropicDiffusion.h.

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

Definition at line 88 of file DGAnisotropicDiffusion.h.

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

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

Definition at line 82 of file DGAnisotropicDiffusion.h.

5.21.4.13 unsigned int DGColumnMassDispersion::_index [private]

Index of the species of interest for this kernel.

Definition at line 82 of file DGColumnMassDispersion.h.

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

Reference to the molecular diffusion material property.

Definition at line 84 of file DGColumnMassDispersion.h.

5.21.4.15 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:

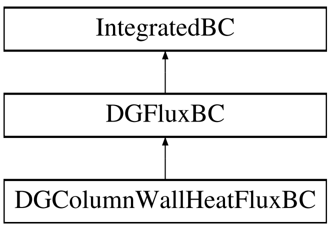
- DGColumnMassDispersion.h

5.22 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.

5.22.1 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.

5.22.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.22.3 Member Function Documentation

5.22.3.1 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.

5.22.3.2 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.

5.22.4 Member Data Documentation

5.22.4.1 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.

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

Reference to the thermal conductivity material property.

Definition at line 82 of file DGColumnWallHeatFluxBC.h.

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

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

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

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

5.22.4.13 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.

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

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

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

Definition at line 85 of file DGFluxBC.h.

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

Definition at line 86 of file DGFluxBC.h.

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

Definition at line 87 of file DGFluxBC.h.

5.22.4.18 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:

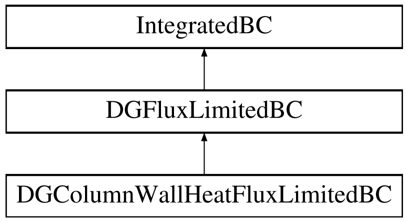
- DGColumnWallHeatFluxBC.h

5.23 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.

5.23.1 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.

5.23.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.23.3 Member Function Documentation

5.23.3.1 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.

5.23.3.2 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.

5.23.4 Member Data Documentation

5.23.4.1 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.

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

Reference to the thermal conductivity material property.

Definition at line 75 of file DGColumnWallHeatFluxLimitedBC.h.

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

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

5.23.4.13 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.

5.23.4.14 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.

5.23.4.15 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.

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

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

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

Definition at line 81 of file DGFluxLimitedBC.h.

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

Definition at line 82 of file DGFluxLimitedBC.h.

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

Definition at line 83 of file DGFluxLimitedBC.h.

5.23.4.20 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:

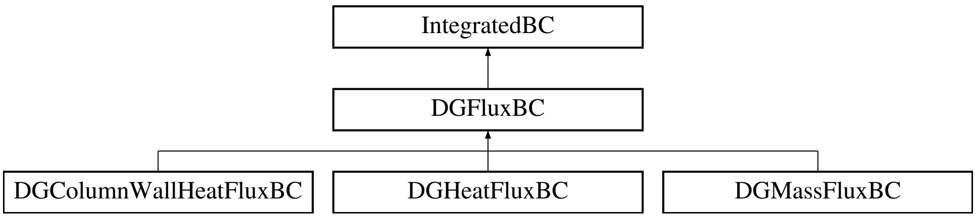
- DGColumnWallHeatFluxLimitedBC.h

5.24 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.

5.24.1 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.

5.24.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.24.3 Member Function Documentation

5.24.3.1 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.

5.24.3.2 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.

5.24.4 Member Data Documentation

5.24.4.1 RealTensorValue DGFluxBC::_Diffusion [protected]

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

5.24.4.2 Real DGFluxBC::_Dxx [protected]

Definition at line 89 of file DGFluxBC.h.

5.24.4.3 Real DGFluxBC::_Dxy [protected]

Definition at line 89 of file DGFluxBC.h.

5.24.4.4 Real DGFluxBC::_Dxz [protected]

Definition at line 89 of file DGFluxBC.h.

5.24.4.5 Real DGFluxBC::_Dyx [protected]

Definition at line 90 of file DGFluxBC.h.

5.24.4.6 Real DGFluxBC::_Dyy [protected]

Definition at line 90 of file DGFluxBC.h.

5.24.4.7 Real DGFluxBC::_Dyz [protected]

Definition at line 90 of file DGFluxBC.h.

5.24.4.8 Real DGFluxBC::_Dzx [protected]

Definition at line 91 of file DGFluxBC.h.

5.24.4.9 Real DGFluxBC::_Dzy [protected]

Definition at line 91 of file DGFluxBC.h.

5.24.4.10 Real DGFluxBC::_Dzz [protected]

Definition at line 91 of file DGFluxBC.h.

5.24.4.11 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.

5.24.4.12 RealVectorValue DGFluxBC::_velocity [protected]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

5.24.4.13 Real DGFluxBC::_vx [protected]

Definition at line 85 of file DGFluxBC.h.

5.24.4.14 Real DGFluxBC::_vy [protected]

Definition at line 86 of file DGFluxBC.h.

5.24.4.15 Real DGFluxBC::_vz [protected]

Definition at line 87 of file DGFluxBC.h.

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

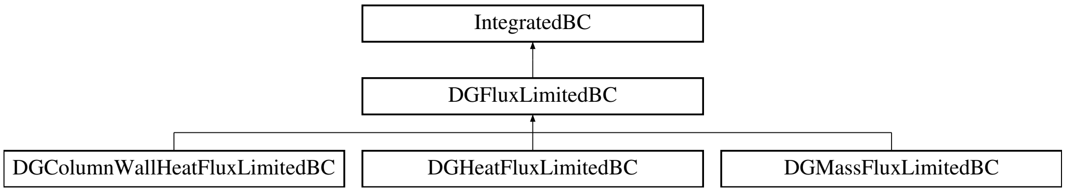
- DGFluxBC.h

5.25 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.

5.25.1 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.

5.25.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.25.3 Member Function Documentation

5.25.3.1 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.

5.25.3.2 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.

5.25.4 Member Data Documentation

5.25.4.1 RealTensorValue DGFluxLimitedBC::_Diffusion [protected]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

5.25.4.2 Real DGFluxLimitedBC::_Dxx [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

5.25.4.3 Real DGFluxLimitedBC::_Dxy [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

5.25.4.4 Real DGFluxLimitedBC::_Dxz [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

5.25.4.5 Real DGFluxLimitedBC::_Dyx [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

5.25.4.6 Real DGFluxLimitedBC::_Dyy [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

5.25.4.7 Real DGFluxLimitedBC::_Dyz [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

5.25.4.8 Real DGFluxLimitedBC::_Dzx [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

5.25.4.9 Real DGFluxLimitedBC::_Dzy [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

5.25.4.10 Real DGFluxLimitedBC::_Dzz [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

5.25.4.11 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.

5.25.4.12 Real DGFluxLimitedBC::_sigma [protected]

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

Definition at line 74 of file DGFluxLimitedBC.h.

5.25.4.13 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.

5.25.4.14 RealVectorValue DGFluxLimitedBC::_velocity [protected]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

5.25.4.15 Real DGFluxLimitedBC::_vx [protected]

Definition at line 81 of file DGFluxLimitedBC.h.

5.25.4.16 Real DGFluxLimitedBC::_vy [protected]

Definition at line 82 of file DGFluxLimitedBC.h.

5.25.4.17 Real DGFluxLimitedBC::_vz [protected]

Definition at line 83 of file DGFluxLimitedBC.h.

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

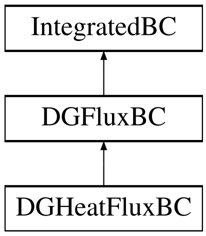
- DGFluxLimitedBC.h

5.26 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.

5.26.1 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.

5.26.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.26.3 Member Function Documentation

5.26.3.1 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.

5.26.3.2 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.

5.26.4 Member Data Documentation

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

Reference to the thermal conductivity material property.

Definition at line 85 of file DGHeatFluxBC.h.

5.26.4.2 RealTensorValue DGFluxBC::_Diffusion `[protected]`, `[inherited]`

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

5.26.4.3 Real DGFluxBC::_Dxx `[protected]`, `[inherited]`

Definition at line 89 of file DGFluxBC.h.

5.26.4.4 Real DGFluxBC::_Dxy `[protected]`, `[inherited]`

Definition at line 89 of file DGFluxBC.h.

5.26.4.5 Real DGFluxBC::_Dxz `[protected]`, `[inherited]`

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Reference to the gas density material property.

Definition at line 83 of file DGHeatFluxBC.h.

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

Reference to the gas heat capacity material property.

Definition at line 84 of file DGHeatFluxBC.h.

5.26.4.14 Real DGHeatFluxBC::_input_temperature [private]

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

Definition at line 80 of file DGHeatFluxBC.h.

5.26.4.15 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.

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

Reference to the velocity material property.

Definition at line 82 of file DGHeatFluxBC.h.

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

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

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

Definition at line 85 of file DGFluxBC.h.

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

Definition at line 86 of file DGFluxBC.h.

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

Definition at line 87 of file DGFluxBC.h.

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

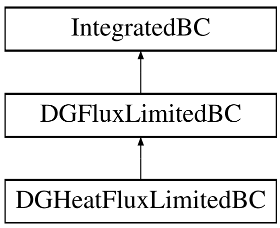
- DGHeatFluxBC.h

5.27 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.

5.27.1 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.

5.27.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.27.3 Member Function Documentation

5.27.3.1 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.

5.27.3.2 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.

5.27.4 Member Data Documentation

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

Reference to the thermal conductivity material property.

Definition at line 78 of file DGHeatFluxLimitedBC.h.

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

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

5.27.4.12 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.

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

Reference to the gas density material property.

Definition at line 76 of file DGHeatFluxLimitedBC.h.

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

Reference to the gas heat capacity material property.

Definition at line 77 of file DGHeatFluxLimitedBC.h.

5.27.4.15 Real DGHeatFluxLimitedBC::_input_temperature [private]

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

Definition at line 73 of file DGHeatFluxLimitedBC.h.

5.27.4.16 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.

5.27.4.17 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.

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

Reference to the velocity material property.

Definition at line 75 of file DGHeatFluxLimitedBC.h.

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

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

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

Definition at line 81 of file DGFluxLimitedBC.h.

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

Definition at line 82 of file DGFluxLimitedBC.h.

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

Definition at line 83 of file DGFluxLimitedBC.h.

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

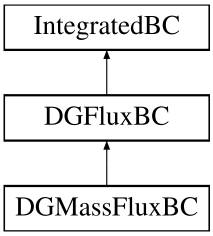
- DGHeatFluxLimitedBC.h

5.28 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.

5.28.1 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.

5.28.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.28.3 Member Function Documentation

5.28.3.1 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.

5.28.3.2 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.

5.28.4 Member Data Documentation

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

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

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

Reference to the dispersion coefficient material property.

Definition at line 88 of file DGMassFluxBC.h.

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

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 89 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 90 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

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

Definition at line 91 of file DGFluxBC.h.

5.28.4.12 unsigned int DGMassFluxBC::_index [private]

Index of the species of interest at the boundary.

Definition at line 87 of file DGMassFluxBC.h.

5.28.4.13 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.

5.28.4.14 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.

5.28.4.15 Real DGMassFluxBC::_input_temperature [private]

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

Definition at line 80 of file DGMassFluxBC.h.

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

Reference to the molecular diffusion material property.

Definition at line 89 of file DGMassFluxBC.h.

5.28.4.17 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.

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

Reference to the velocity material property.

Definition at line 86 of file DGMassFluxBC.h.

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

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

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

Definition at line 85 of file DGFluxBC.h.

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

Definition at line 86 of file DGFluxBC.h.

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

Definition at line 87 of file DGFluxBC.h.

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

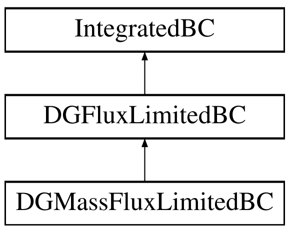
- DGMassFluxBC.h

5.29 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.

5.29.1 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.

5.29.2 Constructor & Destructor Documentation

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

Required constructor for BC objects in MOOSE.

5.29.3 Member Function Documentation

5.29.3.1 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.

5.29.3.2 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.

5.29.4 Member Data Documentation

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

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

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

Reference to the dispersion coefficient material property.

Definition at line 81 of file DGMassFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 85 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 86 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

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

Definition at line 87 of file DGFluxLimitedBC.h.

5.29.4.12 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.

5.29.4.13 unsigned int DGMassFluxLimitedBC::_index [private]

Index of the species of interest at the boundary.

Definition at line 80 of file DGMassFluxLimitedBC.h.

5.29.4.14 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.

5.29.4.15 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.

5.29.4.16 Real DGMassFluxLimitedBC::_input_temperature [private]

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

Definition at line 73 of file DGMassFluxLimitedBC.h.

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

Reference to the molecular diffusion material property.

Definition at line 82 of file DGMassFluxLimitedBC.h.

5.29.4.18 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.

5.29.4.19 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.

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

Reference to the velocity material property.

Definition at line 79 of file DGMassFluxLimitedBC.h.

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

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

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

Definition at line 81 of file DGFluxLimitedBC.h.

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

Definition at line 82 of file DGFluxLimitedBC.h.

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

Definition at line 83 of file DGFluxLimitedBC.h.

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

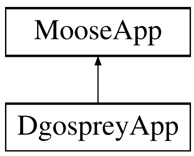
- DGMassFluxLimitedBC.h

5.30 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?)

5.30.1 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.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 DgospreyApp::DgospreyApp (InputParameters *parameters*)

DgospreyApp constructor (required)

5.30.2.2 virtual DgospreyApp::∼DgospreyApp ( ) [virtual]

DgospreyApp destructor (requried)

5.30.3 Member Function Documentation

5.30.3.1 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.

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

Function to the DgospreyApp into MOOSE (required)

5.30.3.3 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

5.31 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_l_I
- std::vector< double > uz_lm1_I
- std::vector< double > uz_lp1_I

    Implicit local slopes (Calculated at Runtime)
- std::vector< double > uz_l_E
- std::vector< double > uz_lm1_E
- std::vector< double > uz_lp1_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)

5.31.1 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.

5.31.2 Member Data Documentation

5.31.2.1 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.

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

Function pointer to executioner (DEFAULT = default_execution)

Definition at line 186 of file finch.h.

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

Centered, explicit coefficients (Calculated at Runtime)

Definition at line 134 of file finch.h.

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

Centered, implicit coefficients (Calculated at Runtime)

Definition at line 133 of file finch.h.

5.31.2.5 bool FINCH_DATA::CheckMass = false

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

Definition at line 117 of file finch.h.

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

Left side, explicit coefficients (Calculated at Runtime)

Definition at line 132 of file finch.h.

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

Left side, implicit coefficients (Calculated at Runtime)

Definition at line 131 of file finch.h.

5.31.2.8 bool FINCH_DATA::CN = true

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

Definition at line 114 of file finch.h.

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

Right side, explicit coefficients (Calculated at Runtime)

Definition at line 136 of file finch.h.

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

Right side, implicit coefficients (Calculated at Runtime)

Definition at line 135 of file finch.h.

5.31.2.11 int FINCH_DATA::d = 0

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

Definition at line 84 of file finch.h.

5.31.2.12 double FINCH_DATA::DIC = 1.0

Initial condition of Dispersion (if constant)

Definition at line 100 of file finch.h.

5.31.2.13 bool FINCH_DATA::Dirichlet = false

Flag to indicate use of Dirichlet or Neumann starting boundary.

Definition at line 116 of file finch.h.

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

Function pointer to discretization (DEFAULT = ospre_discretization)

Definition at line 192 of file finch.h.

5.31.2.15 Matrix<double> FINCH_DATA::Dn

Dispersion Old.

Definition at line 166 of file finch.h.

5.31.2.16 Matrix<double> FINCH_DATA::Dnp1

Dispersion New.

Definition at line 167 of file finch.h.

5.31.2.17 double FINCH_DATA::Do = 1.0

Boundary Value of Dispersion.

Definition at line 105 of file finch.h.

5.31.2.18 double FINCH_DATA::dt = 0.0125

Time step.

Definition at line 85 of file finch.h.

5.31.2.19 double FINCH_DATA::dt_old = 0.0125

Previous time step.

Definition at line 86 of file finch.h.

5.31.2.20 double FINCH_DATA::dz = 0.1

Space step.

Definition at line 88 of file finch.h.

5.31.2.21 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.

5.31.2.22 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.

5.31.2.23 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.

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

Centered, explicit fluxes (Calculated at Runtime)

Definition at line 141 of file finch.h.

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

Centered, implicit fluxes (Calculated at Runtime)

Definition at line 140 of file finch.h.

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

Left side, explicit fluxes (Calculated at Runtime)

Definition at line 139 of file finch.h.

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

Left side, implicit fluxes (Calculated at Runtime)

Definition at line 138 of file finch.h.

5.31.2.28 Matrix<double> FINCH_DATA::Fn

Flux Limiter Old.

Definition at line 175 of file finch.h.

5.31.2.29 Matrix<double> FINCH_DATA::Fnp1

Flux Limiter New.

Definition at line 176 of file finch.h.

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

Right side, explicit fluxes (Calculated at Runtime)

Definition at line 143 of file finch.h.

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

Right side, implicit fluxes (Calculated at Runtime)

Definition at line 142 of file finch.h.

5.31.2.32 Matrix<double> FINCH_DATA::gE

Explicit Side Boundary Conditions.

Definition at line 178 of file finch.h.

5.31.2.33 Matrix<double> FINCH_DATA::gI

Implicit Side Boundary Conditions.

Definition at line 177 of file finch.h.

5.31.2.34 bool FINCH_DATA::Iterative = true

Flag to indicate whether to solve directly, or iteratively.

Definition at line 119 of file finch.h.

5.31.2.35 double FINCH_DATA::kfn = 1.0

Film mass transfer coefficient Old.

Definition at line 108 of file finch.h.

5.31.2.36 double FINCH_DATA::kfnp1 = 1.0

Film mass transfer coefficient New.

Definition at line 109 of file finch.h.

5.31.2.37 double FINCH_DATA::kIC = 1.0

Initial condition of Reaction (if constant)

Definition at line 101 of file finch.h.

5.31.2.38 Matrix<double> FINCH_DATA::kn

Reaction Old.

Definition at line 168 of file finch.h.

5.31.2.39 Matrix<double> FINCH_DATA::knp1

Reaction New.

Definition at line 169 of file finch.h.

5.31.2.40 double FINCH_DATA::ko = 1.0

Boundary Value of Reaction.

Definition at line 106 of file finch.h.

5.31.2.41 double FINCH_DATA::L = 1.0

Total space.

Definition at line 89 of file finch.h.

5.31.2.42 double FINCH_DATA::lambda_E

Boundary Coefficient for Explicit Neumann (Calculated at Runtime)

Definition at line 111 of file finch.h.

5.31.2.43 double FINCH_DATA::lambda_I

Boundary Coefficient for Implicit Neumann (Calculated at Runtime)

Definition at line 110 of file finch.h.

5.31.2.44 int FINCH_DATA::LN = 10

Number of nodes.

Definition at line 113 of file finch.h.

5.31.2.45 int FINCH_DATA::max_iter = 20

Maximum number of iterations allowed.

Definition at line 126 of file finch.h.

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

Explicit lower diagonal matrix elements (Calculated at Runtime)

Definition at line 151 of file finch.h.

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

Implicit lower diagonal matrix elements (Calculated at Runtime)

Definition at line 150 of file finch.h.

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

Explicit diagonal matrix elements (Calculated at Runtime)

Definition at line 149 of file finch.h.

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

Implicit diagonal matrix elements (Calculated at Runtime)

Definition at line 148 of file finch.h.

5.31.2.50 int FINCH_DATA::nl_method = FINCH_Picard

Non-linear solution method - default = FINCH_Picard.

Definition at line 128 of file finch.h.

5.31.2.51 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.

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

Explicit upper diagonal matrix elements (Calculated at Runtime)

Definition at line 147 of file finch.h.

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

Implicit upper diagonal matrix elements (Calculated at Runtime)

Definition at line 146 of file finch.h.

5.31.2.54 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.

5.31.2.55 PICARD_DATA FINCH_DATA::picard_dat

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

Definition at line 202 of file finch.h.

5.31.2.56 PJFNK_DATA FINCH_DATA::pjfnk_dat

Data structure for PJFNK method (more rigours method)

Definition at line 203 of file finch.h.

5.31.2.57 Matrix<double> FINCH_DATA::pres

Current search direction.

Definition at line 181 of file finch.h.

5.31.2.58 Matrix<double> FINCH_DATA::res

Current residual.

Definition at line 180 of file finch.h.

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

Function pointer to reset time (DEFAULT = default_reset)

Definition at line 199 of file finch.h.

5.31.2.60 double FINCH_DATA::RIC = 1.0

Initial condition of the Time Coefficient (if constant)

Definition at line 102 of file finch.h.

5.31.2.61 Matrix<double> FINCH_DATA::Rn

Time Coeff Old.

Definition at line 172 of file finch.h.

5.31.2.62 Matrix<double> FINCH_DATA::Rnp1

Time Coeff New.

Definition at line 173 of file finch.h.

5.31.2.63 double FINCH_DATA::Ro = 1.0

Boundary Value of Time Coefficient.

Definition at line 107 of file finch.h.

5.31.2.64 double FINCH_DATA::s = 1.0

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

Definition at line 90 of file finch.h.

5.31.2.65 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.

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

Function pointer to initial conditions (DEFAULT = default_ic)

Definition at line 187 of file finch.h.

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

Function pointer to set parameters (DEFAULT = default_params)

Definition at line 191 of file finch.h.

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

Function pointer to the postprocesses (DEFAULT = default_postprocess)

Definition at line 198 of file finch.h.

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

Function pointer to preprocesses (DEFAULT = default_preprocess)

Definition at line 189 of file finch.h.

5.31.2.70 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.

5.31.2.71 Matrix<double> FINCH_DATA::Sn

Forcing Function Old.

Definition at line 170 of file finch.h.

5.31.2.72 Matrix<double> FINCH_DATA::Snp1

Forcing Function New.

Definition at line 171 of file finch.h.

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

Function pointer to the solver (DEFAULT = default_solve)

Definition at line 190 of file finch.h.

5.31.2.74 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.

5.31.2.75 double FINCH_DATA::T = 1.0

Total time.

Definition at line 87 of file finch.h.

5.31.2.76 double FINCH_DATA::t = 0.0

Current Time.

Definition at line 91 of file finch.h.

5.31.2.77 double FINCH_DATA::t_old = 0.0

Previous Time.

Definition at line 92 of file finch.h.

5.31.2.78 double FINCH_DATA::tol_abs = 1e-6

Absolute Tolerance for Convergence.

Definition at line 125 of file finch.h.

5.31.2.79 double FINCH_DATA::tol_rel = 1e-6

Relative Tolerance for Convergence.

Definition at line 124 of file finch.h.

5.31.2.80 int FINCH_DATA::total_iter = 0

Total number of iterations made.

Definition at line 127 of file finch.h.

5.31.2.81 Matrix<double> FINCH_DATA::u_star

Conserved Quantity Projected New.

Definition at line 161 of file finch.h.

5.31.2.82 double FINCH_DATA::uAvg = 0.0

Average amount of conserved quantity in domain.

Definition at line 96 of file finch.h.

5.31.2.83 double FINCH_DATA::uAvg_old = 0.0

Old Average amount of conserved quantity.

Definition at line 97 of file finch.h.

5.31.2.84 Matrix<double> FINCH_DATA::ubest

Best found solution if solving iteratively.

Definition at line 162 of file finch.h.

5.31.2.85 double FINCH_DATA::uIC = 0.0

Initial condition of Conserved Quantity (if constant)

Definition at line 98 of file finch.h.

5.31.2.86 Matrix<double> FINCH_DATA::un

Conserved Quantity Old.

Definition at line 159 of file finch.h.

5.31.2.87 Matrix<double> FINCH_DATA::unm1

Conserved Quantity Older.

Definition at line 158 of file finch.h.

5.31.2.88 Matrix<double> FINCH_DATA::unp1

Conserved Quantity New.

Definition at line 160 of file finch.h.

5.31.2.89 double FINCH_DATA::uo = 1.0

Boundary Value of Conserved Quantity.

Definition at line 103 of file finch.h.

5.31.2.90 bool FINCH_DATA::Update = false

Flag to check if the system needs updating.

Definition at line 115 of file finch.h.

5.31.2.91 double FINCH_DATA::uT = 0.0

Total amount of conserved quantity in domain.

Definition at line 94 of file finch.h.

5.31.2.92 double FINCH_DATA::uT_old = 0.0

Old Total amount of conserved quantity.

Definition at line 95 of file finch.h.

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

Definition at line 155 of file finch.h.

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

Definition at line 154 of file finch.h.

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

Definition at line 155 of file finch.h.

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

Definition at line 154 of file finch.h.

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

Explicit local slopes (Calculated at Runtime)

Definition at line 155 of file finch.h.

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

Implicit local slopes (Calculated at Runtime)

Definition at line 154 of file finch.h.

5.31.2.99 double FINCH_DATA::vIC = 1.0

Initial condition of Velocity (if constant)

Definition at line 99 of file finch.h.

5.31.2.100 Matrix<double> FINCH_DATA::vn

Velocity Old.

Definition at line 164 of file finch.h.

5.31.2.101 Matrix<double> FINCH_DATA::vnp1

Velocity New.

Definition at line 165 of file finch.h.

5.31.2.102 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:

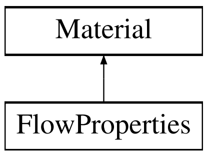
- finch.h

5.32 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.

- virtual void initQpStatefulProperties ()

  Required function override for Stateful Material objects in MOOSE.

Private Attributes

- std::vector< Real > _molecular_weight

  Molecular weights 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.
- const MaterialProperty< Real > & _pellet_diameter

  Coupled material property for the adsorbent pellet diameter.
- const MaterialProperty< Real > & _binder_porosity

  MaterialProperty for the binder porosity.
- const MaterialProperty< Real > & _pore_size

  MaterialProperty for the macropore radius (cm)
- 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.
- MaterialProperty< MIXED_GAS > & _mixed_gas

  MaterialProperty for the MIXED_GAS struct in egret.h.
- MaterialProperty< MIXED_GAS > & _mixed_gas_old

  Old MaterialProperty for the MIXED_GAS struct in egret.h.
- MaterialProperty< std::vector
  < Real > > & _film_transfer

  MaterialProperty for the film mass transfer coeff (cm/hr)
- MaterialProperty< std::vector
  < Real > > & _pore_diffusion

  MaterialProperty for the pore diffusion (cm^2/hr)
- 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.

5.32.1 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.

5.32.2 Constructor & Destructor Documentation

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

Required constructor for objects in MOOSE.

5.32.3 Member Function Documentation

5.32.3.1 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.

5.32.3.2 virtual void FlowProperties::initQpStatefulProperties ( ) [protected], [virtual]

Required function override for Stateful Material objects in MOOSE.

This function is needed because we have to properly initialize our custom objects without having to reinitialize at each compute step. It takes more memory this way, but also prevents segfault errors and helps the kernel run faster after initialization.

5.32.4 Member Data Documentation

5.32.4.1 const MaterialProperty<Real>& FlowProperties::_binder_porosity [private]

MaterialProperty for the binder porosity.

Definition at line 100 of file FlowProperties.h.

5.32.4.2 Real FlowProperties::_column_length [private]

Length of the fixed-bed column (cm)

Definition at line 87 of file FlowProperties.h.

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

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

Definition at line 82 of file FlowProperties.h.

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

Sutherland's reference temperature for each species (K)

Definition at line 84 of file FlowProperties.h.

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

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

Definition at line 83 of file FlowProperties.h.

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

Sutherland's constant for each gas species (K)

Definition at line 85 of file FlowProperties.h.

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

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

Definition at line 105 of file FlowProperties.h.

5.32.4.8 MaterialProperty<std::vector<Real> >& FlowProperties::_film_transfer [private]

MaterialProperty for the film mass transfer coeff (cm/hr)

Definition at line 110 of file FlowProperties.h.

5.32.4.9 Real FlowProperties::_flow_rate [private]

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

Definition at line 86 of file FlowProperties.h.

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

Pointer list to the coupled gases.

Definition at line 116 of file FlowProperties.h.

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

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

Definition at line 90 of file FlowProperties.h.

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

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

Definition at line 92 of file FlowProperties.h.

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

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

Definition at line 93 of file FlowProperties.h.

5.32.4.14 MaterialProperty<Real>& FlowProperties::_gas_viscosity [private]

MaterialProperty for the gas viscosity (g/cm/s)

Definition at line 91 of file FlowProperties.h.

5.32.4.15 MaterialProperty<Real>& FlowProperties::_heat_retardation [private]

MaterialProperty for energy balance retardation coefficient.

Definition at line 103 of file FlowProperties.h.

5.32.4.16 std::vector<unsigned int> FlowProperties::_index [private]

Indices for the gas species in the system.

Definition at line 115 of file FlowProperties.h.

5.32.4.17 const MaterialProperty<Real>& FlowProperties::_inner_dia [private]

Coupled material property for bed inner diameter.

Definition at line 95 of file FlowProperties.h.

5.32.4.18 MaterialProperty< MIXED_GAS >& FlowProperties::_mixed_gas [private]

MaterialProperty for the MIXED_GAS struct in egret.h.

Definition at line 107 of file FlowProperties.h.

5.32.4.19 MaterialProperty< MIXED_GAS >& FlowProperties::_mixed_gas_old [private]

Old MaterialProperty for the MIXED_GAS struct in egret.h.

Definition at line 108 of file FlowProperties.h.

5.32.4.20 MaterialProperty<std::vector<Real> >& FlowProperties::_molecular_diffusion [private]

MaterialProperty for each species' molecular diffusion (cm^2/s)

Definition at line 104 of file FlowProperties.h.

5.32.4.21 std::vector<Real> FlowProperties::_molecular_weight [private]

Molecular weights for each gas species (g/mol)

Definition at line 81 of file FlowProperties.h.

5.32.4.22 const MaterialProperty<Real>& FlowProperties::_pellet_density [private]

Coupled material property for adsorbent pellet density.

Definition at line 97 of file FlowProperties.h.

5.32.4.23 const MaterialProperty<Real>& FlowProperties::_pellet_diameter [private]

Coupled material property for the adsorbent pellet diameter.

Definition at line 99 of file FlowProperties.h.

5.32.4.24 const MaterialProperty<Real>& FlowProperties::_pellet_heat_capacity [private]

Coupled material property for adsorbent heat capacity.

Definition at line 98 of file FlowProperties.h.

5.32.4.25 MaterialProperty<std::vector<Real> >& FlowProperties::_pore_diffusion [private]

MaterialProperty for the pore diffusion (cm^2/hr)

Definition at line 111 of file FlowProperties.h.

5.32.4.26 const MaterialProperty<Real>& FlowProperties::_pore_size [private]

MaterialProperty for the macropore radius (cm)

Definition at line 101 of file FlowProperties.h.

5.32.4.27 const MaterialProperty<Real>& FlowProperties::_porosity [private]

Coupled material property for bed bulk porosity.

Definition at line 96 of file FlowProperties.h.

5.32.4.28 MaterialProperty<std::vector<Real> >& FlowProperties::_retardation [private]

MaterialProperty for each species' retardation coefficient.

Definition at line 106 of file FlowProperties.h.

5.32.4.29 std::vector<VariableValue *> FlowProperties::_solid_conc [private]

Pointer list to the coupled adsorption concentrations.

Definition at line 117 of file FlowProperties.h.

5.32.4.30 std::vector<VariableValue *> FlowProperties::_solid_perturb [private]

Pointer list to the coupled adsorption perturbations.

Definition at line 118 of file FlowProperties.h.

5.32.4.31 VariableValue& FlowProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 113 of file FlowProperties.h.

5.32.4.32 VariableValue& FlowProperties::_total_pressure [private]

Reference to the coupled column pressure.

Definition at line 114 of file FlowProperties.h.

5.32.4.33 MaterialProperty<Real>& FlowProperties::_velocity [private]

MaterialProperty for the linear velocity in the bed (cm/hr)

Definition at line 89 of file FlowProperties.h.

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

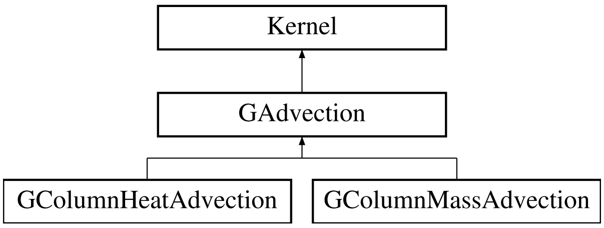
- FlowProperties.h

5.33 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)

5.33.1 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.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 GAdvection::GAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.33.3 Member Function Documentation

5.33.3.1 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.

5.33.3.2 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.

5.33.4 Member Data Documentation

5.33.4.1 RealVectorValue GAdvection::_velocity [protected]

Vector of velocity.

Definition at line 74 of file GAdvection.h.

5.33.4.2 Real GAdvection::_vx [protected]

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

Definition at line 76 of file GAdvection.h.

5.33.4.3 Real GAdvection::_vy [protected]

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

Definition at line 77 of file GAdvection.h.

5.33.4.4 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:

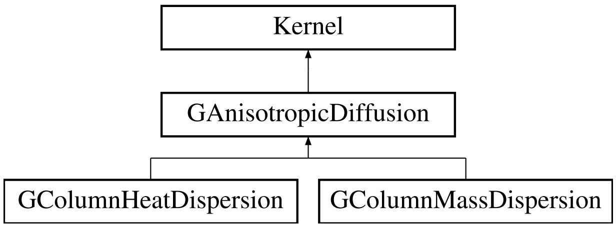
- GAdvection.h

5.34 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

5.34.1 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.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 GAnisotropicDiffusion::GAnisotropicDiffusion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.34.3 Member Function Documentation

5.34.3.1 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.

5.34.3.2 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.

5.34.4 Member Data Documentation

5.34.4.1 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected]

Diffusion tensor matrix parameter.

Definition at line 74 of file GAnisotropicDiffusion.h.

5.34.4.2 Real GAnisotropicDiffusion::_Dxx [protected]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.34.4.3 Real GAnisotropicDiffusion::_Dxy [protected]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.34.4.4 Real GAnisotropicDiffusion::_Dxz [protected]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.34.4.5 Real GAnisotropicDiffusion::_Dyx [protected]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.34.4.6 Real GAnisotropicDiffusion::_Dyy [protected]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.34.4.7 Real GAnisotropicDiffusion::_Dyz [protected]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.34.4.8 Real GAnisotropicDiffusion::_Dzx [protected]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.34.4.9 Real GAnisotropicDiffusion::_Dzy [protected]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.34.4.10 Real GAnisotropicDiffusion::_Dzz [protected]

Definition at line 78 of file GAnisotropicDiffusion.h.

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

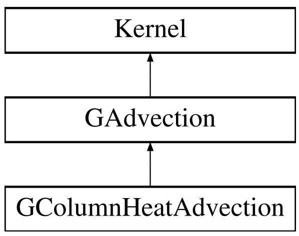
- GAnisotropicDiffusion.h

5.35 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.

5.35.1 Detailed Description

GColumnHeatAdvection class object inherits from GAdvection object.

This class object inherits from the generic GAdvection kernel for use with the corresponding DGColumnHeat-Advection 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.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 GColumnHeatAdvection::GColumnHeatAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.35.3 Member Function Documentation

5.35.3.1 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.

5.35.3.2 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.

5.35.4 Member Data Documentation

5.35.4.1 const MaterialProperty<Real>& GColumnHeatAdvection::_gas_density [private]

Reference to the gas density material property.

Definition at line 75 of file GColumnHeatAdvection.h.

5.35.4.2 const MaterialProperty<Real>& GColumnHeatAdvection::_gas_heat_capacity [private]

Reference to the gas heat capacity material property.

Definition at line 76 of file GColumnHeatAdvection.h.

5.35.4.3 const MaterialProperty<Real>& GColumnHeatAdvection::_vel [private]

Reference to the linear velocity material property.

Definition at line 74 of file GColumnHeatAdvection.h.

5.35.4.4 RealVectorValue GAdvection::_velocity [protected], [inherited]

Vector of velocity.

Definition at line 74 of file GAdvection.h.

5.35.4.5 Real GAdvection::_vx [protected], [inherited]

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

Definition at line 76 of file GAdvection.h.

5.35.4.6 Real GAdvection::_vy [protected], [inherited]

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

Definition at line 77 of file GAdvection.h.

5.35.4.7 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:

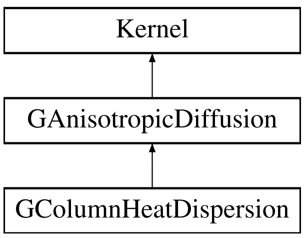
- GColumnHeatAdvection.h

5.36 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.

5.36.1 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.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 GColumnHeatDispersion::GColumnHeatDispersion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.36.3 Member Function Documentation

5.36.3.1 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.

5.36.3.2 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.

5.36.4 Member Data Documentation

5.36.4.1 const MaterialProperty<Real>& GColumnHeatDispersion::_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 73 of file GColumnHeatDispersion.h.

5.36.4.2 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 74 of file GAnisotropicDiffusion.h.

5.36.4.3 Real GAnisotropicDiffusion::_Dxx [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.36.4.4 Real GAnisotropicDiffusion::_Dxy [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.36.4.5 Real GAnisotropicDiffusion::_Dxz [protected], [inherited]

Definition at line 76 of file GAnisotropicDiffusion.h.

5.36.4.6 Real GAnisotropicDiffusion::_Dyx [protected], [inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.36.4.7 Real GAnisotropicDiffusion::_Dyy [protected], [inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.36.4.8 Real GAnisotropicDiffusion::_Dyz [protected], [inherited]

Definition at line 77 of file GAnisotropicDiffusion.h.

5.36.4.9 Real GAnisotropicDiffusion::_Dzx [protected], [inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.36.4.10 Real GAnisotropicDiffusion::_Dzy [protected], [inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

5.36.4.11 Real GAnisotropicDiffusion::_Dzz [protected], [inherited]

Definition at line 78 of file GAnisotropicDiffusion.h.

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

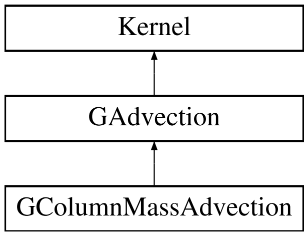
- GColumnHeatDispersion.h

5.37 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.

5.37.1 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.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 GColumnMassAdvection::GColumnMassAdvection (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.37.3 Member Function Documentation

5.37.3.1 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.

5.37.3.2 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.

5.37.4 Member Data Documentation

5.37.4.1 const MaterialProperty<Real>& GColumnMassAdvection::_vel [private]

Reference to the linear velocity material property.

Definition at line 72 of file GColumnMassAdvection.h.

5.37.4.2 RealVectorValue GAdvection::_velocity [protected], [inherited]

Vector of velocity.

Definition at line 74 of file GAdvection.h.

5.37.4.3 Real GAdvection::_vx [protected], [inherited]

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

Definition at line 76 of file GAdvection.h.

5.37.4.4 Real GAdvection::_vy [protected], [inherited]

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

Definition at line 77 of file GAdvection.h.

5.37.4.5 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:

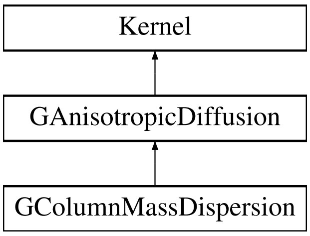
- GColumnMassAdvection.h

5.38 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.

5.38.1 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.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 GColumnMassDispersion::GColumnMassDispersion (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.38.3 Member Function Documentation

5.38.3.1 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.

5.38.3.2 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.

5.38.4 Member Data Documentation

5.38.4.1 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 74 of file GAnisotropicDiffusion.h.

5.38.4.2 const MaterialProperty<std::vector<Real> >& GColumnMassDispersion::_dispersion `[private]`

Reference to the dispersion material property.

Definition at line 74 of file GColumnMassDispersion.h.

5.38.4.3 Real GAnisotropicDiffusion::_Dxx `[protected], [inherited]`

Definition at line 76 of file GAnisotropicDiffusion.h.

5.38.4.4 Real GAnisotropicDiffusion::_Dxy `[protected], [inherited]`

Definition at line 76 of file GAnisotropicDiffusion.h.

5.38.4.5 Real GAnisotropicDiffusion::_Dxz `[protected], [inherited]`

Definition at line 76 of file GAnisotropicDiffusion.h.

5.38.4.6 Real GAnisotropicDiffusion::_Dyx `[protected], [inherited]`

Definition at line 77 of file GAnisotropicDiffusion.h.

5.38.4.7 Real GAnisotropicDiffusion::_Dyy `[protected], [inherited]`

Definition at line 77 of file GAnisotropicDiffusion.h.

5.38.4.8 Real GAnisotropicDiffusion::_Dyz `[protected], [inherited]`

Definition at line 77 of file GAnisotropicDiffusion.h.

5.38.4.9 Real GAnisotropicDiffusion::_Dzx `[protected], [inherited]`

Definition at line 78 of file GAnisotropicDiffusion.h.

5.38.4.10 Real GAnisotropicDiffusion::_Dzy `[protected], [inherited]`

Definition at line 78 of file GAnisotropicDiffusion.h.

5.38.4.11 Real GAnisotropicDiffusion::_Dzz `[protected], [inherited]`

Definition at line 78 of file GAnisotropicDiffusion.h.

5.38.4.12 unsigned int GColumnMassDispersion::_index `[private]`

Index of the species of interest for this kernel.

Definition at line 73 of file GColumnMassDispersion.h.

5.38.4.13 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

5.39 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.

5.39.1 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.

5.39.2 Member Data Documentation

5.39.2.1 double GCR_DATA::alpha

Inner iteration step size.

Definition at line 345 of file lark.h.

5.39.2.2 double GCR_DATA::bestres

Best found residual norm of the linear system.

Definition at line 352 of file lark.h.

5.39.2.3 Matrix<double> GCR_DATA::bestx

Best found solution to the linear system.

Definition at line 357 of file lark.h.

5.39.2.4 double GCR_DATA::beta

Outer iteration step size.

Definition at line 346 of file lark.h.

5.39.2.5 bool GCR_DATA::breakdown = false

Boolean to determine if a step has failed.

Definition at line 343 of file lark.h.

5.39.2.6 std::vector<Matrix<double> > GCR_DATA::c

Vector span for updating r.

Definition at line 362 of file lark.h.

5.39.2.7 Matrix<double> GCR_DATA::c_temp

Temporary c vector to be updated.

Definition at line 359 of file lark.h.

5.39.2.8 int GCR_DATA::iter_inner = 0

Number of inner iterations taken.

Definition at line 341 of file lark.h.

5.39.2.9 int GCR_DATA::iter_outer = 0

Number of outer iterations taken.

Definition at line 340 of file lark.h.

5.39.2.10 int GCR_DATA::maxit = 0

Maximum allowable outer iterations.

Definition at line 339 of file lark.h.

5.39.2.11 bool GCR_DATA::Output = true

True = print messages to the console.

Definition at line 354 of file lark.h.

5.39.2.12 Matrix<double> GCR_DATA::r

Residual Vector.

Definition at line 358 of file lark.h.

5.39.2.13 double GCR_DATA::relres

Relative residual norm for linear system.

Definition at line 350 of file lark.h.

5.39.2.14 double GCR_DATA::relres_base

Initial residual norm of the linear system.

Definition at line 351 of file lark.h.

5.39.2.15 double GCR_DATA::res

Absolute residual norm for linear system.

Definition at line 349 of file lark.h.

5.39.2.16 int GCR_DATA::restart = -1

Restart parameter for outer iterations - default = 20.

Definition at line 338 of file lark.h.

5.39.2.17 double GCR_DATA::tol_abs = 1e-6

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

Definition at line 348 of file lark.h.

5.39.2.18 double GCR_DATA::tol_rel = 1e-6

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

Definition at line 347 of file lark.h.

5.39.2.19 int GCR_DATA::total_iter = 0

Total number of iterations taken.

Definition at line 342 of file lark.h.

5.39.2.20 OPTRANS_DATA GCR_DATA::transpose_dat

Data structure for Operator Transposition.

Definition at line 364 of file lark.h.

5.39.2.21 std::vector<Matrix<double> > GCR_DATA::u

Vector span for updating x.

Definition at line 361 of file lark.h.

5.39.2.22 Matrix<double> GCR_DATA::u_temp

Temporary u vector to be updated.

Definition at line 360 of file lark.h.

5.39.2.23 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

5.40 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.

5.40.1 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.

5.40.2 Member Data Documentation

5.40.2.1 ARNOLDI_DATA GMRESLP_DATA::arnoldi_dat

Data structure for the kyrlov subspace.

Definition at line 167 of file lark.h.

5.40.2.2 double GMRESLP_DATA::bestres

Best found residual norm of the linear system.

Definition at line 159 of file lark.h.

5.40.2.3 Matrix<double> GMRESLP_DATA::bestx

Best found solution to the linear system.

Definition at line 164 of file lark.h.

5.40.2.4 int GMRESLP_DATA::iter = 0

Number of iterations needed for convergence.

Definition at line 151 of file lark.h.

5.40.2.5 int GMRESLP_DATA::maxit = 0

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

Definition at line 150 of file lark.h.

5.40.2.6 bool GMRESLP_DATA::Output = true

True = print messages to console.

Definition at line 161 of file lark.h.

5.40.2.7 Matrix<double> GMRESLP_DATA::r

Residual vector for the linear system.

Definition at line 165 of file lark.h.

5.40.2.8 double GMRESLP_DATA::relres

Relative residual norm of the linear system.

Definition at line 157 of file lark.h.

5.40.2.9 double GMRESLP_DATA::relres_base

Initial residual norm of the linear system.

Definition at line 158 of file lark.h.

5.40.2.10 double GMRESLP_DATA::res

Absolution redisual norm of the linear system.

Definition at line 156 of file lark.h.

5.40.2.11 int GMRESLP_DATA::restart = -1

Restart parameter - default = min(vector_size,20)

Definition at line 149 of file lark.h.

5.40.2.12 int GMRESLP_DATA::steps = 0

Total number of gmres iterations and krylov iterations.

Definition at line 152 of file lark.h.

5.40.2.13 double GMRESLP_DATA::tol_abs = 1e-6

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

Definition at line 155 of file lark.h.

5.40.2.14 double GMRESLP_DATA::tol_rel = 1e-6

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

Definition at line 154 of file lark.h.

5.40.2.15 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

5.41 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.

5.41.1 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.

5.41.2 Member Data Documentation

5.41.2.1 Matrix<double> GMRESR_DATA::arg

Argument matrix passed between preconditioner and iterator.

Definition at line 391 of file lark.h.

5.41.2.2 double GMRESR_DATA::gcr_abs_tol = 1e-6

Absolute outer residual tolerance.

Definition at line 389 of file lark.h.

5.41.2.3 GCR_DATA GMRESR_DATA::gcr_dat

Data structure for the outer GCR steps.

Definition at line 393 of file lark.h.

5.41.2.4 int GMRESR_DATA::gcr_maxit = 0

Number of GCR iterations.

Definition at line 376 of file lark.h.

5.41.2.5 bool GMRESR_DATA::GCR_Output = true

True = print GCR messages.

Definition at line 384 of file lark.h.

5.41.2.6 double GMRESR_DATA::gcr_rel_tol = 1e-6

Relative outer residual tolerance.

Definition at line 388 of file lark.h.

5.41.2.7 int GMRESR_DATA::gcr_restart = -1

Number of GCR restarts (default = 20, max = N)

Definition at line 375 of file lark.h.

5.41.2.8 GMRESRP_DATA GMRESR_DATA::gmres_dat

Data structure for the inner GMRES steps.

Definition at line 394 of file lark.h.

5.41.2.9 int GMRESR_DATA::gmres_maxit = 1

Number of GMRES iterations (max = 5, default = 1)

Definition at line 378 of file lark.h.

5.41.2.10 bool GMRESR_DATA::GMRES_Output = false

True = print GMRES messages.

Definition at line 385 of file lark.h.

5.41.2.11 int GMRESR_DATA::gmres_restart = -1

Number of GMRES restarts (max = 20)

Definition at line 377 of file lark.h.

5.41.2.12 double GMRESR_DATA::gmres_tol = 0.1

Tolerance relative to GCR iterations.

Definition at line 387 of file lark.h.

5.41.2.13 int GMRESR_DATA::iter_inner

Total GMRES iterations.

Definition at line 382 of file lark.h.

5.41.2.14 int GMRESR_DATA::iter_outer

Total GCR iterations.

Definition at line 381 of file lark.h.

5.41.2.15 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.

5.41.2.16 const void∗ GMRESR_DATA::matvec_data

Data structure for the user's matvec function.

Definition at line 401 of file lark.h.

5.41.2.17 int GMRESR_DATA::N

Dimension of the linear system.

Definition at line 379 of file lark.h.

5.41.2.18 const void∗ GMRESR_DATA::term_precon

Data structure for the user's terminal preconditioner.

Definition at line 402 of file lark.h.

5.41.2.19 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.

5.41.2.20 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

5.42 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

5.42.1 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.

5.42.2 Member Data Documentation

5.42.2.1 double GMRESRP_DATA::bestres

Best found residual norm of the linear system.

Definition at line 190 of file lark.h.

5.42.2.2 Matrix<double> GMRESRP_DATA::bestx

Best found solution to the linear system.

Definition at line 195 of file lark.h.

5.42.2.3 std::vector< double > GMRESRP_DATA::e0

(k+1 x 1) Normalized vector with residual info

Definition at line 203 of file lark.h.

5.42.2.4 std::vector< double > GMRESRP_DATA::e0_bar

(k+1 x 1) Factorized normal vector

Definition at line 204 of file lark.h.

5.42.2.5 std::vector< std::vector< double > > GMRESRP_DATA::H

(k+1 x k) upper Hessenberg storage matrix

Definition at line 200 of file lark.h.

5.42.2.6 std::vector< std::vector< double > > GMRESRP_DATA::H_bar

(k+1 x k) Factorized matrix

Definition at line 201 of file lark.h.

5.42.2.7 int GMRESRP_DATA::iter_inner = 0

Total number of inner iterations.

Definition at line 182 of file lark.h.

5.42.2.8 int GMRESRP_DATA::iter_outer = 0

Total number of outer iterations.

Definition at line 181 of file lark.h.

5.42.2.9 int GMRESRP_DATA::iter_total = 0

Total number of overall iterations.

Definition at line 183 of file lark.h.

5.42.2.10 int GMRESRP_DATA::maxit = 0

Maximum allowable outer iterations.

Definition at line 180 of file lark.h.

5.42.2.11 bool GMRESRP_DATA::Output = true

True = print messages to console.

Definition at line 192 of file lark.h.

5.42.2.12 Matrix<double> GMRESRP_DATA::r

Residual vector for the linear system.

Definition at line 196 of file lark.h.

5.42.2.13 double GMRESRP_DATA::relres

Relative residual norm for linear system.

Definition at line 188 of file lark.h.

5.42.2.14 double GMRESRP_DATA::relres_base

Initial residual norm of the linear system.

Definition at line 189 of file lark.h.

5.42.2.15 double GMRESRP_DATA::res

Absolute residual norm for linear system.

Definition at line 187 of file lark.h.

5.42.2.16 int GMRESRP_DATA::restart = -1

Restart parameter - default = min(20,vector_size)

Definition at line 179 of file lark.h.

5.42.2.17 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.

5.42.2.18 double GMRESRP_DATA::tol_abs = 1e-6

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

Definition at line 186 of file lark.h.

5.42.2.19 double GMRESRP_DATA::tol_rel = 1e-6

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

Definition at line 185 of file lark.h.

5.42.2.20 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.

5.42.2.21 std::vector< Matrix<double> > GMRESRP_DATA::Vk

(N x k) orthonormal vector basis

Definition at line 198 of file lark.h.

5.42.2.22 Matrix<double> GMRESRP_DATA::w

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

Definition at line 206 of file lark.h.

5.42.2.23 Matrix<double> GMRESRP_DATA::x

Current solution to the linear system.

Definition at line 194 of file lark.h.

5.42.2.24 std::vector< double > GMRESRP_DATA::y

(k x 1) Vector search direction

Definition at line 202 of file lark.h.

5.42.2.25 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

5.43 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 PIo

  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.

5.43.1 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.

5.43.2 Member Data Documentation

5.43.2.1 std::vector<double> GPAST_DATA::gama_inf

Infinite dilution activities.

Definition at line 129 of file magpie.h.

5.43.2.2 double GPAST_DATA::He

Henry's Coefficient (mol/kg/kPa)

Definition at line 127 of file magpie.h.

5.43.2.3 double GPAST_DATA::PIo

Pure component spreading pressures (mol/kg)

Definition at line 131 of file magpie.h.

5.43.2.4 std::vector<double> GPAST_DATA::po

Pure component reference state pressures (kPa)

Definition at line 132 of file magpie.h.

5.43.2.5 double GPAST_DATA::poi

Reference state pressures solved for using Recover eval GPAST.

Definition at line 133 of file magpie.h.

5.43.2.6 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.

5.43.2.7 double GPAST_DATA::q

Amount adsorbed for each component (mol/kg)

Definition at line 128 of file magpie.h.

5.43.2.8 double GPAST_DATA::qo

Pure component capacities (mol/kg)

Definition at line 130 of file magpie.h.

5.43.2.9 double GPAST_DATA::x

Adsorbed mole fraction.

Definition at line 125 of file magpie.h.

5.43.2.10 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

5.44 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))

5.44.1 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.

5.44.2 Member Data Documentation

5.44.2.1 std::vector<double> GSTA_DATA::dHo

Enthalpies for each site (J/mol)

Definition at line 102 of file magpie.h.

5.44.2.2 std::vector<double> GSTA_DATA::dSo

Entropies for each site (J/(K∗mol))

Definition at line 103 of file magpie.h.

5.44.2.3 int GSTA_DATA::m

Number of parameters in the GSTA isotherm.

Definition at line 101 of file magpie.h.

5.44.2.4 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

5.45 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.

5.45.1 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.

5.45.2 Member Data Documentation

5.45.2.1 std::vector<GMRESRP_DATA> KMS_DATA::gmres_in

Data structures for each recursion level.

Definition at line 431 of file lark.h.

5.45.2.2 GMRESRP_DATA KMS_DATA::gmres_out

Data structure for the outer steps.

Definition at line 430 of file lark.h.

5.45.2.3 int KMS_DATA::inner_iter = 0

Number of inner steps taken.

Definition at line 419 of file lark.h.

5.45.2.4 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.

5.45.2.5 int KMS_DATA::level = 0

Current level in the recursion.

Definition at line 415 of file lark.h.

5.45.2.6 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.

5.45.2.7 const void* KMS_DATA::matvec_data

Data structure for the user's matvec function.

Definition at line 438 of file lark.h.

5.45.2.8 int KMS_DATA::max_level = 0

Maximum allowable recursion levels (Default = 0 -> GMRES, Max = 5)

Definition at line 416 of file lark.h.

5.45.2.9 int KMS_DATA::maxit = 0

Maximum allowable iterations for the outer steps.

Definition at line 418 of file lark.h.

5.45.2.10 double KMS_DATA::outer_abstol = 1e-6

Absolute residual tolerance for outer steps (Default = 1e-6)

Definition at line 424 of file lark.h.

5.45.2.11 int KMS_DATA::outer_iter = 0

Number of outer steps taken.

Definition at line 420 of file lark.h.

5.45.2.12 double KMS_DATA::outer_reltol = 1e-6

Relative residual tolerance for outer steps (Default = 1e-6)

Definition at line 423 of file lark.h.

5.45.2.13 bool KMS_DATA::Output_in = false

True = Print the inner steps residuals.

Definition at line 428 of file lark.h.

5.45.2.14 bool KMS_DATA::Output_out = true

True = Print the outer steps residuals.

Definition at line 427 of file lark.h.

5.45.2.15 int KMS_DATA::restart = -1

Restart parameter for the outer iterates (Default = 20, Max = N)

Definition at line 417 of file lark.h.

5.45.2.16 const void* KMS_DATA::term_precon

Data structure for the user's terminal preconditioner.

Definition at line 439 of file lark.h.

5.45.2.17 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.

5.45.2.18 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:

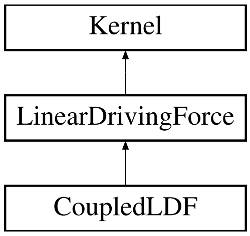
- lark.h

5.46 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.

5.46.1 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.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 LinearDrivingForce::LinearDrivingForce (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.46.3 Member Function Documentation

5.46.3.1 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.

5.46.3.2 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.

5.46.4 Member Data Documentation

5.46.4.1 Real LinearDrivingForce::_coef [protected]

Coefficient for the strength or rate of the driving force.

Definition at line 77 of file LinearDrivingForce.h.

5.46.4.2 Real LinearDrivingForce::_driving_value [protected]

Value the coupled variable is driving towards.

Definition at line 78 of file LinearDrivingForce.h.

5.46.4.3 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.

5.46.4.4 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:

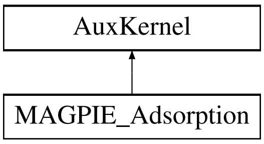
- LinearDrivingForce.h

5.47 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.

5.47.1 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.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 MAGPIE_Adsorption::MAGPIE_Adsorption (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.47.3 Member Function Documentation

5.47.3.1 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.

5.47.4 Member Data Documentation

5.47.4.1 unsigned int MAGPIE_Adsorption::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 76 of file MAGPIE_Adsorption.h.

5.47.4.2 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:

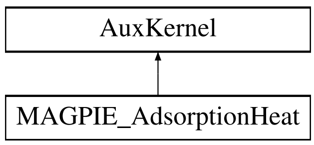
- MAGPIE_Adsorption.h

5.48 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)

5.48.1 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.

5.48.2 Constructor & Destructor Documentation

5.48.2.1 MAGPIE_AdsorptionHeat::MAGPIE_AdsorptionHeat (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.48.3 Member Function Documentation

5.48.3.1 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.

5.48.4 Member Data Documentation

5.48.4.1 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.

5.48.4.2 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.

5.48.4.3 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:

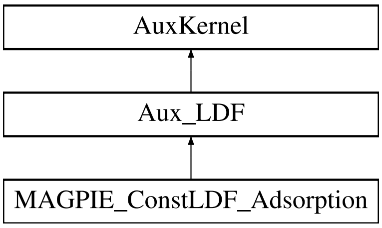
- MAGPIE_AdsorptionHeat.h

5.49 MAGPIE_ConstLDF_Adsorption Class Reference

MAGPIE_ConstLDF class inherits from AuxKernel.

`#include <MAGPIE_ConstLDF_Adsorption.h>`

Inheritance diagram for MAGPIE_ConstLDF_Adsorption:



Public Member Functions

- MAGPIE_ConstLDF_Adsorption (const InputParameters &parameters)

    Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

    Required MOOSE function override.

Protected Attributes

- Real _ldf_coef
- Real _driving_value

    Value of the driving force coefficient.

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_ConstLDF class inherits from AuxKernel.

This class object inherits from Aux_LDF to calculated the adsorption of an aux variable based on a constant linear driving force parameter and a MAGPIE simulation. The MAGPIE simulation is used to override the driving value of the base class at every iteration, thus coupling the kinetics to the transport problem. NOTE: This coupling should be done loosely to avoid poor convergence behavior between the multiple scales of the problem.

Definition at line 65 of file MAGPIE_ConstLDF_Adsorption.h.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 MAGPIE_ConstLDF_Adsorption::MAGPIE_ConstLDF_Adsorption (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.49.3 Member Function Documentation

5.49.3.1 virtual Real MAGPIE_ConstLDF_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.

Reimplemented from Aux_LDF.

5.49.4 Member Data Documentation

5.49.4.1 Real Aux_LDF::_driving_value [protected], [inherited]

Value of the driving force coefficient.

Definition at line 71 of file Aux_LDF.h.

5.49.4.2 unsigned int MAGPIE_ConstLDF_Adsorption::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 78 of file MAGPIE_ConstLDF_Adsorption.h.

5.49.4.3 Real Aux_LDF::_ldf_coef [protected], [inherited]

Definition at line 70 of file Aux_LDF.h.

5.49.4.4 const MaterialProperty< MAGPIE_DATA >& MAGPIE_ConstLDF_Adsorption::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 79 of file MAGPIE_ConstLDF_Adsorption.h.

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

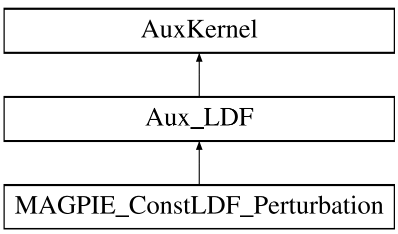
- MAGPIE_ConstLDF_Adsorption.h

5.50 MAGPIE_ConstLDF_Perturbation Class Reference

MAGPIE_ConstLDF class inherits from AuxKernel.

`#include <MAGPIE_ConstLDF_Perturbation.h>`

Inheritance diagram for MAGPIE_ConstLDF_Perturbation:



Public Member Functions

- MAGPIE_ConstLDF_Perturbation (const InputParameters &parameters)

    Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

  Required MOOSE function override.

Protected Attributes

- Real _ldf_coef
- Real _driving_value

  Value of the driving force coefficient.

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.

5.50.1 Detailed Description

MAGPIE_ConstLDF class inherits from AuxKernel.

This class object inherits from Aux_LDF to calculated the adsorption perturbation of an aux variable based on a constant linear driving force parameter and a MAGPIE simulation. The MAGPIE simulation is used to override the driving value of the base class at every iteration, thus coupling the kinetics to the transport problem. NOTE: This coupling should be done loosely to avoid poor convergence behavior between the multiple scales of the problem.

Definition at line 65 of file MAGPIE_ConstLDF_Perturbation.h.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 MAGPIE_ConstLDF_Perturbation::MAGPIE_ConstLDF_Perturbation (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.50.3 Member Function Documentation

5.50.3.1 virtual Real MAGPIE_ConstLDF_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.

Reimplemented from Aux_LDF.

5.50.4 Member Data Documentation

5.50.4.1 Real Aux_LDF::_driving_value [protected], [inherited]

Value of the driving force coefficient.

Definition at line 71 of file Aux_LDF.h.

5.50.4.2 unsigned int MAGPIE_ConstLDF_Perturbation::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 78 of file MAGPIE_ConstLDF_Perturbation.h.

5.50.4.3 Real Aux_LDF::_ldf_coef [protected], [inherited]

Definition at line 70 of file Aux_LDF.h.

5.50.4.4 const MaterialProperty< MAGPIE_DATA >& MAGPIE_ConstLDF_Perturbation::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 79 of file MAGPIE_ConstLDF_Perturbation.h.

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

- MAGPIE_ConstLDF_Perturbation.h

5.51 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

5.51.1 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.

5.51.2 Member Data Documentation

5.51.2.1 std::vector<GPAST_DATA> MAGPIE_DATA::gpast_dat

Definition at line 168 of file magpie.h.

5.51.2.2 std::vector<GSTA_DATA> MAGPIE_DATA::gsta_dat

Definition at line 166 of file magpie.h.

5.51.2.3 std::vector<mSPD_DATA> MAGPIE_DATA::mspd_dat

Definition at line 167 of file magpie.h.

5.51.2.4 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:

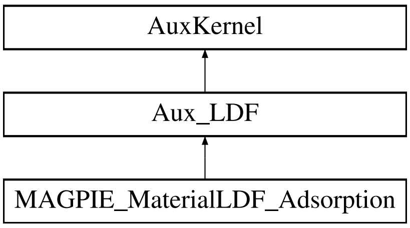
- magpie.h

5.52 MAGPIE_MaterialLDF_Adsorption Class Reference

MAGPIE_MaterialLDF_Adsorption class inherits from AuxKernel.

`#include <MAGPIE_MaterialLDF_Adsorption.h>`

Inheritance diagram for MAGPIE_MaterialLDF_Adsorption:



Public Member Functions

- MAGPIE_MaterialLDF_Adsorption (const InputParameters &parameters)

    Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

    Required MOOSE function override.

Protected Attributes

- Real _ldf_coef
- Real _driving_value

    Value of the driving force coefficient.

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.
- const MaterialProperty< Real > & _pellet_diameter

    Coupled material property for the adsorbent pellet diameter.
- const MaterialProperty< Real > & _porosity

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

    MaterialProperty for the binder porosity.
- const MaterialProperty< Real > & _crystal_radius

    MaterialProperty for the crystal radius (um)
- const MaterialProperty< Real > & _pellet_density

    MaterialProperty for the pellet density.

- const MaterialProperty
  < std::vector< Real > > & _film_transfer

  MaterialProperty for the film mass transfer coeff (cm/hr)
- const MaterialProperty
  < std::vector< Real > > & _pore_diffusion

  MaterialProperty for the pore diffusion (cm^2/hr)
- const MaterialProperty
  < std::vector< Real > > & _surface_diffusion

  MaterialProperty for the surface diffusion (um^2/hr)

5.52.1 Detailed Description

MAGPIE_MaterialLDF_Adsorption class inherits from AuxKernel.

This class object inherits from Aux_LDF to calculated the adsorption of an aux variable based on a variable linear driving force parameter and a MAGPIE simulation. The LDF parameter is calculated based on the values of parameters in material property files and the MAGPIE simulation is used to override the driving value of the base class at every iteration, thus coupling the kinetics to the transport problem. NOTE: This coupling should be done loosely to avoid poor convergence behavior between the multiple scales of the problem.

Definition at line 66 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.2 Constructor & Destructor Documentation

5.52.2.1 MAGPIE_MaterialLDF_Adsorption::MAGPIE_MaterialLDF_Adsorption (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.52.3 Member Function Documentation

5.52.3.1 virtual Real MAGPIE_MaterialLDF_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.

Reimplemented from Aux_LDF.

5.52.4 Member Data Documentation

5.52.4.1 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Adsorption::_binder_porosity [private]

MaterialProperty for the binder porosity.

Definition at line 84 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.2 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Adsorption::_crystal_radius [private]

MaterialProperty for the crystal radius (um)

Definition at line 85 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.3 Real Aux_LDF::_driving_value [protected], [inherited]

Value of the driving force coefficient.

Definition at line 71 of file Aux_LDF.h.

5.52.4.4 const MaterialProperty<std::vector<Real> >& MAGPIE_MaterialLDF_Adsorption::_film_transfer [private]

MaterialProperty for the film mass transfer coeff (cm/hr)

Definition at line 88 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.5 unsigned int MAGPIE_MaterialLDF_Adsorption::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 79 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.6 Real Aux_LDF::_ldf_coef [protected], [inherited]

Definition at line 70 of file Aux_LDF.h.

5.52.4.7 const MaterialProperty< MAGPIE_DATA >& MAGPIE_MaterialLDF_Adsorption::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 80 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.8 const MaterialProperty<Real >& MAGPIE_MaterialLDF_Adsorption::_pellet_density [private]

MaterialProperty for the pellet density.

Definition at line 86 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.9 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Adsorption::_pellet_diameter [private]

Coupled material property for the adsorbent pellet diameter.

Definition at line 82 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.10 const MaterialProperty<std::vector<Real> >& MAGPIE_MaterialLDF_Adsorption::_pore_diffusion [private]

MaterialProperty for the pore diffusion (cm^2/hr)

Definition at line 89 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.11 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Adsorption::_porosity [private]

Coupled material property for bed bulk porosity.

Definition at line 83 of file MAGPIE_MaterialLDF_Adsorption.h.

5.52.4.12 const MaterialProperty<std::vector<Real> >& MAGPIE_MaterialLDF_Adsorption::_surface_diffusion [private]

MaterialProperty for the surface diffusion (um^2/hr)

Definition at line 90 of file MAGPIE_MaterialLDF_Adsorption.h.

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

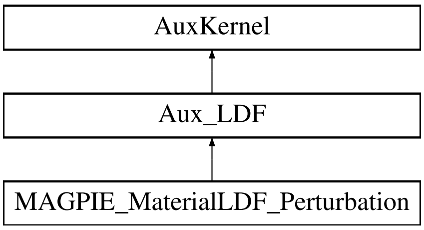
- MAGPIE_MaterialLDF_Adsorption.h

5.53 MAGPIE_MaterialLDF_Perturbation Class Reference

MAGPIE_MaterialLDF_Perturbation class inherits from AuxKernel.

`#include <MAGPIE_MaterialLDF_Perturbation.h>`

Inheritance diagram for MAGPIE_MaterialLDF_Perturbation:



Public Member Functions

- MAGPIE_MaterialLDF_Perturbation (const InputParameters &parameters)

    Standard MOOSE public constructor.

Protected Member Functions

- virtual Real computeValue ()

    Required MOOSE function override.

Protected Attributes

- Real _ldf_coef
- Real _driving_value

    Value of the driving force coefficient.

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.
- const MaterialProperty< Real > & _pellet_diameter

    Coupled material property for the adsorbent pellet diameter.
- const MaterialProperty< Real > & _porosity

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

    MaterialProperty for the binder porosity.
- const MaterialProperty< Real > & _crystal_radius

    MaterialProperty for the crystal radius (um)
- const MaterialProperty< Real > & _pellet_density

    MaterialProperty for the pellet density.
- const MaterialProperty
  < std::vector< Real > > & _film_transfer

    MaterialProperty for the film mass transfer coeff (cm/hr)
- const MaterialProperty
  < std::vector< Real > > & _pore_diffusion

    MaterialProperty for the pore diffusion (cm^2/hr)
- const MaterialProperty
  < std::vector< Real > > & _surface_diffusion

    MaterialProperty for the surface diffusion (um^2/hr)

5.53.1 Detailed Description

MAGPIE_MaterialLDF_Perturbation class inherits from AuxKernel.

This class object inherits from Aux_LDF to calculated the adsorption perturbation of an aux variable based on a variable linear driving force parameter and a MAGPIE simulation. The LDF parameter is calculated based on the values of parameters in material property files and the MAGPIE simulation is used to override the driving value of the base class at every iteration, thus coupling the kinetics to the transport problem. NOTE: This coupling should be done loosely to avoid poor convergence behavior between the multiple scales of the problem.

Definition at line 66 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.2 Constructor & Destructor Documentation

5.53.2.1 MAGPIE_MaterialLDF_Perturbation::MAGPIE_MaterialLDF_Perturbation (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.53.3 Member Function Documentation

5.53.3.1 virtual Real MAGPIE_MaterialLDF_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.

Reimplemented from Aux_LDF.

5.53.4 Member Data Documentation

5.53.4.1 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Perturbation::_binder_porosity [private]

MaterialProperty for the binder porosity.

Definition at line 84 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.2 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Perturbation::_crystal_radius [private]

MaterialProperty for the crystal radius (um)

Definition at line 85 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.3 Real Aux_LDF::_driving_value [protected], [inherited]

Value of the driving force coefficient.

Definition at line 71 of file Aux_LDF.h.

5.53.4.4 const MaterialProperty<std::vector<Real> >& MAGPIE_MaterialLDF_Perturbation::_film_transfer [private]

MaterialProperty for the film mass transfer coeff (cm/hr)

Definition at line 88 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.5 unsigned int MAGPIE_MaterialLDF_Perturbation::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 79 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.6 Real Aux_LDF::_ldf_coef [protected], [inherited]

Definition at line 70 of file Aux_LDF.h.

5.53.4.7 const MaterialProperty< MAGPIE_DATA >& MAGPIE_MaterialLDF_Perturbation::_magpie_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 80 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.8 const MaterialProperty<Real >& MAGPIE_MaterialLDF_Perturbation::_pellet_density [private]

MaterialProperty for the pellet density.

Definition at line 86 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.9 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Perturbation::_pellet_diameter [private]

Coupled material property for the adsorbent pellet diameter.

Definition at line 82 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.10 const MaterialProperty<std::vector<Real> >& MAGPIE_MaterialLDF_Perturbation::_pore_diffusion [private]

MaterialProperty for the pore diffusion (cm^2/hr)

Definition at line 89 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.11 const MaterialProperty<Real>& MAGPIE_MaterialLDF_Perturbation::_porosity [private]

Coupled material property for bed bulk porosity.

Definition at line 83 of file MAGPIE_MaterialLDF_Perturbation.h.

5.53.4.12 const MaterialProperty<std::vector<Real> >& MAGPIE_MaterialLDF_Perturbation::_surface_diffusion [private]

MaterialProperty for the surface diffusion (um^2/hr)

Definition at line 90 of file MAGPIE_MaterialLDF_Perturbation.h.

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

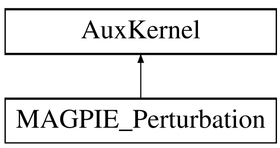
- MAGPIE_MaterialLDF_Perturbation.h

5.54 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.

5.54.1 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.

5.54.2 Constructor & Destructor Documentation

5.54.2.1 MAGPIE_Perturbation::MAGPIE_Perturbation (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.54.3 Member Function Documentation

5.54.3.1 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.

5.54.4 Member Data Documentation

5.54.4.1 unsigned int MAGPIE_Perturbation::_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 85 of file MAGPIE_Perturbation.h.

5.54.4.2 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:

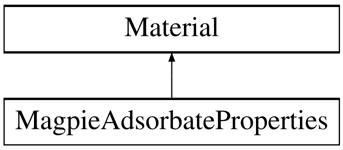
- MAGPIE_Perturbation.h

5.55 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.
- virtual void initQpStatefulProperties ()

  Required function override for Stateful 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.
- MaterialProperty< MAGPIE_DATA > & _magpie_dat_old

  Old MaterialProperty object to hold the MAGPIE_DATA structure and all relavent information.

5.55.1 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.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 MagpieAdsorbateProperties::MagpieAdsorbateProperties (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.55.3 Member Function Documentation

5.55.3.1 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.

5.55.3.2 virtual void MagpieAdsorbateProperties::initQpStatefulProperties ( ) [protected], [virtual]

Required function override for Stateful Material objects in MOOSE.

This function is needed because we have to properly initialize our custom objects without having to reinitialize at each compute step. It takes more memory this way, but also prevents segfault errors and helps the kernel run faster after initialization.

5.55.4 Member Data Documentation

5.55.4.1 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_1 [private]

List of the site 1 enthalpies for each gas species.

Definition at line 91 of file MagpieAdsorbateProperties.h.

5.55.4.2 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_2 [private]

List of the site 2 enthalpies for each gas species.

Definition at line 92 of file MagpieAdsorbateProperties.h.

5.55.4.3 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_3 [private]

List of the site 3 enthalpies for each gas species.

Definition at line 93 of file MagpieAdsorbateProperties.h.

5.55.4.4 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_4 [private]

List of the site 4 enthalpies for each gas species.

Definition at line 94 of file MagpieAdsorbateProperties.h.

5.55.4.5 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_5 [private]

List of the site 5 enthalpies for each gas species.

Definition at line 95 of file MagpieAdsorbateProperties.h.

5.55.4.6 std::vector<Real> MagpieAdsorbateProperties::_enthalpy_6 [private]

List of the site 6 enthalpies for each gas species.

Definition at line 96 of file MagpieAdsorbateProperties.h.

5.55.4.7 std::vector<Real> MagpieAdsorbateProperties::_entropy_1 [private]

List of the site 1 entropies for each gas species.

Definition at line 98 of file MagpieAdsorbateProperties.h.

5.55.4.8 std::vector<Real> MagpieAdsorbateProperties::_entropy_2 [private]

List of the site 2 entropies for each gas species.

Definition at line 99 of file MagpieAdsorbateProperties.h.

5.55.4.9 std::vector<Real> MagpieAdsorbateProperties::_entropy_3 [private]

List of the site 3 entropies for each gas species.

Definition at line 100 of file MagpieAdsorbateProperties.h.

5.55.4.10 std::vector<Real> MagpieAdsorbateProperties::_entropy_4 [private]

List of the site 4 entropies for each gas species.

Definition at line 101 of file MagpieAdsorbateProperties.h.

5.55.4.11 std::vector<Real> MagpieAdsorbateProperties::_entropy_5 [private]

List of the site 5 entropies for each gas species.

Definition at line 102 of file MagpieAdsorbateProperties.h.

5.55.4.12 std::vector<Real> MagpieAdsorbateProperties::_entropy_6 [private]

List of the site 6 entropies for each gas species.

Definition at line 103 of file MagpieAdsorbateProperties.h.

5.55.4.13 std::vector<VariableValue *> MagpieAdsorbateProperties::_gas_conc [private]

Pointer list to the coupled gases.

Definition at line 84 of file MagpieAdsorbateProperties.h.

5.55.4.14 std::vector<VariableValue *> MagpieAdsorbateProperties::_gas_conc_old [private]

Pointer list to the old states of coupled gases.

Definition at line 85 of file MagpieAdsorbateProperties.h.

5.55.4.15 std::vector<unsigned int> MagpieAdsorbateProperties::_index [private]

Indices for the gas species in the system.

Definition at line 81 of file MagpieAdsorbateProperties.h.

5.55.4.16 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 109 of file MagpieAdsorbateProperties.h.

5.55.4.17 MaterialProperty< MAGPIE_DATA >& MagpieAdsorbateProperties::_magpie_dat_old [private]

Old MaterialProperty object to hold the MAGPIE_DATA structure and all relavent information.

This object is required to be created in order to use the stateful properties, which is how we intialize our custom objects in MOOSE correctly.

Definition at line 114 of file MagpieAdsorbateProperties.h.

5.55.4.18 std::vector<Real> MagpieAdsorbateProperties::_max_capacity [private]

List of the maximum adsorption capacities of each gas species.

Definition at line 88 of file MagpieAdsorbateProperties.h.

5.55.4.19 std::vector<Real> MagpieAdsorbateProperties::_molar_volume [private]

List of the van der Waal's molar volumes of each species.

Definition at line 89 of file MagpieAdsorbateProperties.h.

5.55.4.20 std::vector<int> MagpieAdsorbateProperties::_num_sites [private]

List of the number of sites each gas species' isotherm contains.

Definition at line 87 of file MagpieAdsorbateProperties.h.

5.55.4.21 VariableValue& MagpieAdsorbateProperties::_temperature [private]

Reference to the coupled column temperature.

Definition at line 82 of file MagpieAdsorbateProperties.h.

5.55.4.22 VariableValue& MagpieAdsorbateProperties::_total_pressure [private]

Reference to the coupled column pressure.

Definition at line 83 of file MagpieAdsorbateProperties.h.

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

- MagpieAdsorbateProperties.h

5.56 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.

5.56.1 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.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 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.

5.56.2.2 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.

5.56.2.3 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.

5.56.2.4 template<class T > Matrix< T >::∼Matrix ( )

Default destructor for clearing out memory.

Definition at line 302 of file macaw.h.

5.56.3 Member Function Documentation

5.56.3.1 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.

5.56.3.2 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.

5.56.3.3 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.

5.56.3.4 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.

5.56.3.5 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.

5.56.3.6 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.

5.56.3.7 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.

5.56.3.8 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.

5.56.3.9 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.

5.56.3.10 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.

5.56.3.11 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().

5.56.3.12 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.

5.56.3.13 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.

5.56.3.14 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.

5.56.3.15 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().

5.56.3.16 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.

5.56.3.17 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().

5.56.3.18 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().

5.56.3.19 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.

5.56.3.20 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.

5.56.3.21 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.

5.56.3.22 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.

5.56.3.23 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.

5.56.3.24 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.

5.56.3.25 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.

5.56.3.26 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().

5.56.3.27 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.

5.56.3.28 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.

5.56.3.29 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.

5.56.3.30 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.

5.56.3.31 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.

5.56.3.32 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.

5.56.3.33 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.

5.56.3.34 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.

5.56.3.35 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.

5.56.3.36 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.

5.56.3.37 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.

5.56.3.38 template<class T > void Matrix< T >::rowShrink ( )

Function to delete the last row of this matrix.

Definition at line 1728 of file macaw.h.

5.56.3.39 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.

5.56.3.40 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.

5.56.3.41 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.

5.56.3.42 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.

5.56.3.43 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.

5.56.3.44 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.

5.56.3.45 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.

5.56.3.46 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.

5.56.3.47 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.

5.56.3.48 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.

5.56.3.49 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.

5.56.3.50 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.

5.56.3.51 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.

5.56.3.52 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.

5.56.4 Member Data Documentation

5.56.4.1 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().

5.56.4.2 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 >::lower-Hessenberg2Triangular(), 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 >::upperTriangular-Solve().

5.56.4.3 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

5.57 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.

5.57.1 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.

5.57.2 Member Data Documentation

5.57.2.1 Matrix<double> MIXED_GAS::binary_diffusion

Calculated: Tensor matrix of binary gas diffusivities (cm^2/s)

Definition at line 136 of file egret.h.

5.57.2.2 double MIXED_GAS::char_length

Given: Characteristic Length (cm)

Definition at line 126 of file egret.h.

5.57.2.3 bool MIXED_GAS::CheckMolefractions = true

Given: True = Check Molefractions for errors.

Definition at line 120 of file egret.h.

5.57.2.4 double MIXED_GAS::gas_temperature

Given: Gas temperature (K)

Definition at line 124 of file egret.h.

5.57.2.5 double MIXED_GAS::kinematic_viscosity

Calculated: Kinematic viscosity (cm^2/s)

Definition at line 132 of file egret.h.

5.57.2.6 std::vector<double> MIXED_GAS::molefraction

Given: Gas molefractions of each species (-)

Definition at line 127 of file egret.h.

5.57.2.7 int MIXED_GAS::N

Given: Total number of gas species.

Definition at line 119 of file egret.h.

5.57.2.8 double MIXED_GAS::Reynolds

Calculated: Value of the Reynold's number (-)

Definition at line 135 of file egret.h.

5.57.2.9 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.

5.57.2.10 double MIXED_GAS::total_density

Calculated: Total gas density (g/cm^3) {use RE3}.

Definition at line 130 of file egret.h.

5.57.2.11 double MIXED_GAS::total_dyn_vis

Calculated: Total dynamic viscosity (g/cm/s)

Definition at line 131 of file egret.h.

5.57.2.12 double MIXED_GAS::total_molecular_weight

Calculated: Total molecular weight (g/mol)

Definition at line 133 of file egret.h.

5.57.2.13 double MIXED_GAS::total_pressure

Given: Total gas pressure (kPa)

Definition at line 123 of file egret.h.

5.57.2.14 double MIXED_GAS::total_specific_heat

Calculated: Total specific heat (J/g/K)

Definition at line 134 of file egret.h.

5.57.2.15 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

5.58 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.

5.58.1 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.

5.58.2 Member Data Documentation

5.58.2.1 double mSPD_DATA::eMax

Maximum lateral interaction energy (J/mol)

Definition at line 114 of file magpie.h.

5.58.2.2 std::vector<double> mSPD_DATA::eta

Binary interaction parameter matrix (i,j)

Definition at line 115 of file magpie.h.

5.58.2.3 double mSPD_DATA::gama

Activity coefficient calculated from mSPD.

Definition at line 116 of file magpie.h.

5.58.2.4 double mSPD_DATA::s

Area shape factor.

Definition at line 112 of file magpie.h.

5.58.2.5 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

5.59 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.

5.59.1 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.

5.59.2 Member Data Documentation

5.59.2.1 Matrix<double> NUM_JAC_DATA::dxj

Vector of perturbed x values.

Definition at line 574 of file lark.h.

5.59.2.2 double NUM_JAC_DATA::eps = sqrt(DBL_EPSILON)

Perturbation value.

Definition at line 571 of file lark.h.

5.59.2.3 Matrix<double> NUM_JAC_DATA::Fx

Vector of function evaluations at x.

Definition at line 572 of file lark.h.

5.59.2.4 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

5.60 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.

5.60.1 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.

5.60.2 Member Data Documentation

5.60.2.1 Matrix<double> OPTRANS_DATA::Ai

The ith column vector of the user's linear operator.

Definition at line 327 of file lark.h.

5.60.2.2 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

5.61 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.

5.61.1 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.

5.61.2 Member Data Documentation

5.61.2.1 double PCG_DATA::alpha

Step size for new solution.

Definition at line 222 of file lark.h.

5.61.2.2 Matrix<double> PCG_DATA::Ap

Result of matrix-vector multiplication.

Definition at line 240 of file lark.h.

5.61.2.3 double PCG_DATA::bestres

Best found residual norm.

Definition at line 229 of file lark.h.

5.61.2.4 Matrix<double> PCG_DATA::bestx

Best found solution to the linear system.

Definition at line 234 of file lark.h.

5.61.2.5 double PCG_DATA::beta

Step size for new search direction.

Definition at line 223 of file lark.h.

5.61.2.6 int PCG_DATA::iter = 0

Actual number of iterations taken.

Definition at line 220 of file lark.h.

5.61.2.7 int PCG_DATA::maxit = 0

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

Definition at line 219 of file lark.h.

5.61.2.8 bool PCG_DATA::Output = true

True = print messages to console.

Definition at line 231 of file lark.h.

5.61.2.9 Matrix<double> PCG_DATA::p

Search direction.

Definition at line 239 of file lark.h.

5.61.2.10 Matrix<double> PCG_DATA::r

Residual vector for the linear system.

Definition at line 235 of file lark.h.

5.61.2.11 Matrix<double> PCG_DATA::r_old

Previous residual vector.

Definition at line 236 of file lark.h.

5.61.2.12 double PCG_DATA::relres

Relative residual norm.

Definition at line 227 of file lark.h.

5.61.2.13 double PCG_DATA::relres_base

Initial residual norm.

Definition at line 228 of file lark.h.

5.61.2.14 double PCG_DATA::res

Absolute residual norm.

Definition at line 226 of file lark.h.

5.61.2.15 double PCG_DATA::tol_abs = 1e-6

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

Definition at line 225 of file lark.h.

5.61.2.16 double PCG_DATA::tol_rel = 1e-6

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

Definition at line 224 of file lark.h.

5.61.2.17 Matrix<double> PCG_DATA::x

Current solution to the linear system.

Definition at line 233 of file lark.h.

5.61.2.18 Matrix<double> PCG_DATA::z

Preconditioned residual vector (result of precon function)

Definition at line 237 of file lark.h.

5.61.2.19 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

5.62 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.

5.62.1 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.

5.62.2 Member Data Documentation

5.62.2.1 double PICARD_DATA::bestres

Best found residual norm.

Definition at line 459 of file lark.h.

5.62.2.2 Matrix<double> PICARD_DATA::bestx

Best found solution vector.

Definition at line 464 of file lark.h.

5.62.2.3 int PICARD_DATA::iter = 0

Actual number of iterations.

Definition at line 452 of file lark.h.

5.62.2.4 int PICARD_DATA::maxit = 0

Maximum allowable iterations - default = min(3∗vec_size,1000)

Definition at line 451 of file lark.h.

5.62.2.5 bool PICARD_DATA::Output = true

True = print messages to console.

Definition at line 461 of file lark.h.

5.62.2.6 Matrix<double> PICARD_DATA::r

Residual of the non-linear system.

Definition at line 465 of file lark.h.

5.62.2.7 double PICARD_DATA::relres

Relative residual norm of the iterate.

Definition at line 457 of file lark.h.

5.62.2.8 double PICARD_DATA::relres_base

Initial residual norm.

Definition at line 458 of file lark.h.

5.62.2.9 double PICARD_DATA::res

Residual norm of the iterate.

Definition at line 456 of file lark.h.

5.62.2.10 double PICARD_DATA::tol_abs = 1e-6

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

Definition at line 455 of file lark.h.

5.62.2.11 double PICARD_DATA::tol_rel = 1e-6

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

Definition at line 454 of file lark.h.

5.62.2.12 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

5.63 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.

5.63.1 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.

5.63.2 Member Data Documentation

5.63.2.1 BACKTRACK_DATA PJFNK_DATA::backtrack_dat

Data structure for the Backtracking Linesearch algorithm.

Definition at line 550 of file lark.h.

5.63.2.2 Matrix<double> PJFNK_DATA::bestx

Best found solution vector to the non-linear system.

Definition at line 538 of file lark.h.

5.63.2.3 BiCGSTAB_DATA PJFNK_DATA::bicgstab_dat

Data structure for the BiCGSTAB method.

Definition at line 543 of file lark.h.

5.63.2.4 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.

5.63.2.5 CGS_DATA PJFNK_DATA::cgs_dat

Data structure for the CGS method.

Definition at line 544 of file lark.h.

5.63.2.6 double PJFNK_DATA::eps =sqrt(DBL_EPSILON)

Value of epsilon used jacvec - default = sqrt(DBL_EPSILON)

Definition at line 527 of file lark.h.

5.63.2.7 Matrix<double> PJFNK_DATA::F

Stored fuction evaluation at x (also the residual)

Definition at line 534 of file lark.h.

5.63.2.8 int PJFNK_DATA::fun_call = 0

Actual number of function calls made.

Definition at line 515 of file lark.h.

5.63.2.9 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.

5.63.2.10 Matrix<double> PJFNK_DATA::Fv

Stored function evaluation at x+eps∗v.

Definition at line 535 of file lark.h.

5.63.2.11 GCR_DATA PJFNK_DATA::gcr_dat

Data structure for the GCR method.

Definition at line 546 of file lark.h.

5.63.2.12 GMRESLP_DATA PJFNK_DATA::gmreslp_dat

Data structure for the GMRESLP method.

Definition at line 541 of file lark.h.

5.63.2.13 GMRESR_DATA PJFNK_DATA::gmresr_dat

Data structure for the GMRESR method.

Definition at line 547 of file lark.h.

5.63.2.14 GMRESRP_DATA PJFNK_DATA::gmresrp_dat

Data structure for the GMRESRP method.

Definition at line 545 of file lark.h.

5.63.2.15 int PJFNK_DATA::l_iter = 0

Number of linear iterations.

Definition at line 514 of file lark.h.

5.63.2.16 bool PJFNK_DATA::L_Output = false

True = print Linear messages to console.

Definition at line 530 of file lark.h.

5.63.2.17 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.

5.63.2.18 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.

5.63.2.19 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.

5.63.2.20 bool PJFNK_DATA::LineSearch = false

True = use Backtracking Linesearch for global convergence.

Definition at line 531 of file lark.h.

5.63.2.21 double PJFNK_DATA::nl_bestres

Best found residual norm.

Definition at line 526 of file lark.h.

5.63.2.22 int PJFNK_DATA::nl_iter = 0

Number of non-linear iterations.

Definition at line 513 of file lark.h.

5.63.2.23 int PJFNK_DATA::nl_maxit = 0

Maximum allowable non-linear steps.

Definition at line 516 of file lark.h.

5.63.2.24 bool PJFNK_DATA::NL_Output = true

True = print PJFNK messages to console.

Definition at line 529 of file lark.h.

5.63.2.25 double PJFNK_DATA::nl_relres

Relative residual for the non-linear system.

Definition at line 524 of file lark.h.

5.63.2.26 double PJFNK_DATA::nl_res

Absolute redidual norm for the non-linear system.

Definition at line 523 of file lark.h.

5.63.2.27 double PJFNK_DATA::nl_res_base

Initial residual norm for the non-linear system.

Definition at line 525 of file lark.h.

5.63.2.28 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.

5.63.2.29 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.

5.63.2.30 PCG_DATA PJFNK_DATA::pcg_dat

Data structure for the PCG method.

Definition at line 542 of file lark.h.

5.63.2.31 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.

5.63.2.32 const void∗ PJFNK_DATA::precon_data

Data structure pointer for user's preconditioning data.

Definition at line 557 of file lark.h.

5.63.2.33 const void∗ PJFNK_DATA::res_data

Data structure pointer for user's residual data.

Definition at line 555 of file lark.h.

5.63.2.34 Matrix<double> PJFNK_DATA::v

Stored vector of x+eps∗v.

Definition at line 536 of file lark.h.

5.63.2.35 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

5.64 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 (-)

5.64.1 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.

5.64.2 Member Data Documentation

5.64.2.1 double PURE_GAS::density

Calculated: gas densities (g/cm^3) {use RE3}.

Definition at line 107 of file egret.h.

5.64.2.2 double PURE_GAS::dynamic_viscosity

Calculated: dynamic viscosities (g/cm/s)

Definition at line 106 of file egret.h.

5.64.2.3 double PURE_GAS::molecular_diffusion

Calculated: molecular diffusivities (cm^2/s)

Definition at line 105 of file egret.h.

5.64.2.4 double PURE_GAS::molecular_weight

Given: molecular weights (g/mol)

Definition at line 98 of file egret.h.

5.64.2.5 double PURE_GAS::Schmidt

Calculated: Value of the Schmidt number (-)

Definition at line 108 of file egret.h.

5.64.2.6 double PURE_GAS::specific_heat

Given: Specific heat of the gas (J/g/K)

Definition at line 102 of file egret.h.

5.64.2.7 double PURE_GAS::Sutherland_Const

Given: Sutherland's Constant (K)

Definition at line 100 of file egret.h.

5.64.2.8 double PURE_GAS::Sutherland_Temp

Given: Sutherland's Reference Temperature (K)

Definition at line 99 of file egret.h.

5.64.2.9 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

5.65 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)

5.65.1 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.

5.65.2 Member Data Documentation

5.65.2.1 double SCOPSOWL_DATA::binder_fraction

Volume of binder per total volume of pellet (-)

Definition at line 116 of file scopsowl.h.

5.65.2.2 double SCOPSOWL_DATA::binder_poresize

Nominal radius of the binder pores (cm)

Definition at line 118 of file scopsowl.h.

5.65.2.3 double SCOPSOWL_DATA::binder_porosity

Volume of pores per volume of binder (-)

Definition at line 117 of file scopsowl.h.

5.65.2.4 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.

5.65.2.5 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.

5.65.2.6 int SCOPSOWL_DATA::coord_macro

Coordinate system for large pellet.

Definition at line 95 of file scopsowl.h.

5.65.2.7 int SCOPSOWL_DATA::coord_micro

Coordinate system for small crystal (if any)

Definition at line 96 of file scopsowl.h.

5.65.2.8 double SCOPSOWL_DATA::crystal_radius

Nominal radius of the crystal - microscale domain (um)

Definition at line 113 of file scopsowl.h.

5.65.2.9 bool SCOPSOWL_DATA::DirichletBC = false

True = Dirichlet BC; False = Neumann BC.

Definition at line 121 of file scopsowl.h.

5.65.2.10 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.

5.65.2.11 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.

5.65.2.12 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.

5.65.2.13 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.

5.65.2.14 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.

5.65.2.15 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.

5.65.2.16 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.

5.65.2.17 double SCOPSOWL_DATA::gas_temperature

Gas phase temperature (K)

Definition at line 111 of file scopsowl.h.

5.65.2.18 double SCOPSOWL_DATA::gas_velocity

Superficial Gas Velocity arount pellet (cm/s)

Definition at line 109 of file scopsowl.h.

5.65.2.19 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.

5.65.2.20 int SCOPSOWL_DATA::level = 2

Level of coupling between the different scales (default = 2)

Definition at line 97 of file scopsowl.h.

5.65.2.21 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.

5.65.2.22 bool SCOPSOWL_DATA::NonLinear = true

True = Non-linear solver; False = Linear solver.

Definition at line 122 of file scopsowl.h.

5.65.2.23 FILE* SCOPSOWL_DATA::OutputFile

Output file pointer to the output file for postprocesses.

Definition at line 126 of file scopsowl.h.

5.65.2.24 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.

5.65.2.25 double SCOPSOWL_DATA::pellet_density

Mass of the pellet per volume of pellet (kg/L)

Definition at line 119 of file scopsowl.h.

5.65.2.26 double SCOPSOWL_DATA::pellet_radius

Nominal radius of the pellet - macroscale domain (cm)

Definition at line 112 of file scopsowl.h.

5.65.2.27 bool SCOPSOWL_DATA::Print2Console = true

True = results to console; False = no printing.

Definition at line 105 of file scopsowl.h.

5.65.2.28 bool SCOPSOWL_DATA::Print2File = true

True = results to .txt; False = no printing.

Definition at line 104 of file scopsowl.h.

5.65.2.29 double SCOPSOWL_DATA::sim_time

Stopping time for the simulation (hrs)

Definition at line 98 of file scopsowl.h.

5.65.2.30 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.

5.65.2.31 bool SCOPSOWL_DATA::SurfDiff = true

True = includes SKUA simulation if Heterogeneous; False = only uses MAGPIE.

Definition at line 106 of file scopsowl.h.

5.65.2.32 double SCOPSOWL_DATA::t

Current time of the simulations (hrs)

Definition at line 100 of file scopsowl.h.

5.65.2.33 double SCOPSOWL_DATA::t_counter = 0.0

Counter for the time output.

Definition at line 101 of file scopsowl.h.

5.65.2.34 double SCOPSOWL_DATA::t_old

Old time of the simulations (hrs)

Definition at line 99 of file scopsowl.h.

5.65.2.35 double SCOPSOWL_DATA::t_print

Print output at every t_print time (hrs)

Definition at line 102 of file scopsowl.h.

5.65.2.36 std::vector<double> SCOPSOWL_DATA::tempy

Temporary place holder for gas mole fractions in other locations (-)

Definition at line 124 of file scopsowl.h.

5.65.2.37 double SCOPSOWL_DATA::total_pressure

Gas phase total pressure (kPa)

Definition at line 110 of file scopsowl.h.

5.65.2.38 unsigned long int SCOPSOWL_DATA::total_steps

Running total of all calculation steps.

Definition at line 94 of file scopsowl.h.

5.65.2.39 const void* SCOPSOWL_DATA::user_data

Data structure for users info to calculate parameters.

Definition at line 133 of file scopsowl.h.

5.65.2.40 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

5.66 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.

5.66.1 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.

5.66.2 Member Data Documentation

5.66.2.1 double SCOPSOWL_PARAM_DATA::activation_energy

Activation energy for surface diffusion (J/mol)

Definition at line 69 of file scopsowl.h.

5.66.2.2 bool SCOPSOWL_PARAM_DATA::Adsorbable

True = species can adsorb; False = species cannot adsorb.

Definition at line 75 of file scopsowl.h.

5.66.2.3 double SCOPSOWL_PARAM_DATA::affinity

Affinity parameter used in empirical adjustments (-)

Definition at line 72 of file scopsowl.h.

5.66.2.4 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.

5.66.2.5 double SCOPSOWL_PARAM_DATA::dq_dco

Boundary value of adsorption slope for Dirichelt BCs (L/kg)

Definition at line 64 of file scopsowl.h.

5.66.2.6 double SCOPSOWL_PARAM_DATA::film_transfer

Value for constant film mass transfer (cm/hr)

Definition at line 67 of file scopsowl.h.

5.66.2.7 double SCOPSOWL_PARAM_DATA::pore_diffusion

Value for constant pore diffusion (cm^2/hr)

Definition at line 66 of file scopsowl.h.

5.66.2.8 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.

5.66.2.9 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.

5.66.2.10 double SCOPSOWL_PARAM_DATA::qIntegralAvg

Integral average of adsorption over the entire pellet (mol/kg)

Definition at line 56 of file scopsowl.h.

5.66.2.11 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.

5.66.2.12 double SCOPSOWL_PARAM_DATA::qo

Boundary value of adsorption if using Dirichlet BCs (mol/kg)

Definition at line 62 of file scopsowl.h.

5.66.2.13 Matrix<double> SCOPSOWL_PARAM_DATA::Qst

Heat of adsorption for all nodes (J/mol)

Definition at line 49 of file scopsowl.h.

5.66.2.14 Matrix<double> SCOPSOWL_PARAM_DATA::Qst_old

Old Heat of adsorption for all nodes (J/mol)

Definition at line 50 of file scopsowl.h.

5.66.2.15 double SCOPSOWL_PARAM_DATA::QstAvg

Integral average heat of adsorption (J/mol)

Definition at line 59 of file scopsowl.h.

5.66.2.16 double SCOPSOWL_PARAM_DATA::QstAvg_old

Old integral average heat of adsorption (J/mol)

Definition at line 60 of file scopsowl.h.

5.66.2.17 double SCOPSOWL_PARAM_DATA::Qsto

Boundary value of adsorption heat if using Dirichlet BCs (J/mol)

Definition at line 63 of file scopsowl.h.

5.66.2.18 double SCOPSOWL_PARAM_DATA::ref_diffusion

Reference state surface diffusivity (um^2/hr)

Definition at line 70 of file scopsowl.h.

5.66.2.19 double SCOPSOWL_PARAM_DATA::ref_pressure

Definition at line 73 of file scopsowl.h.

5.66.2.20 double SCOPSOWL_PARAM_DATA::ref_temperature

Reference temperature for empirical adjustments (K)

Definition at line 71 of file scopsowl.h.

5.66.2.21 std::string SCOPSOWL_PARAM_DATA::speciesName

String to hold the name of each species.

Definition at line 77 of file scopsowl.h.

5.66.2.22 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

5.67 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.

5.67.1 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.

5.67.2 Member Data Documentation

5.67.2.1 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.

5.67.2.2 int SKUA_DATA::coord

Used to determine the coordinates of the problem.

Definition at line 87 of file skua.h.

5.67.2.3 bool SKUA_DATA::DirichletBC = true

True = Dirichlet BC; False = Neumann BC.

Definition at line 101 of file skua.h.

5.67.2.4 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.

5.67.2.5 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.

5.67.2.6 std::vector<FINCH_DATA> SKUA_DATA::finch_dat

Data structure for adsorption kinetics (see finch.h)

Definition at line 111 of file skua.h.

5.67.2.7 MIXED_GAS* SKUA_DATA::gas_dat

Pointer to the MIXED_GAS data structure (see egret.h)

Definition at line 110 of file skua.h.

5.67.2.8 double SKUA_DATA::gas_velocity

Superficial Gas Velocity arount pellet (cm/s)

Definition at line 98 of file skua.h.

5.67.2.9 MAGPIE_DATA SKUA_DATA::magpie_dat

Data structure for adsorption equilibria (see magpie.h)

Definition at line 109 of file skua.h.

5.67.2.10 bool SKUA_DATA::NonLinear = true

True = Non-linear solver; False = Linear solver.

Definition at line 102 of file skua.h.

5.67.2.11 FILE* SKUA_DATA::OutputFile

Output file pointer to the output file.

Definition at line 105 of file skua.h.

5.67.2.12 std::vector<SKUA_PARAM> SKUA_DATA::param_dat

Data structure for SKUA specific parameters.

Definition at line 112 of file skua.h.

5.67.2.13 double SKUA_DATA::pellet_radius

Nominal radius of the pellet/crystal (um)

Definition at line 99 of file skua.h.

5.67.2.14 bool SKUA_DATA::Print2Console = true

True = results to console; False = no printing.

Definition at line 96 of file skua.h.

5.67.2.15 bool SKUA_DATA::Print2File = true

True = results to .txt; False = no printing.

Definition at line 95 of file skua.h.

5.67.2.16 double SKUA_DATA::qTn

Old total amounts adsorbed (mol/kg)

Definition at line 93 of file skua.h.

5.67.2.17 double SKUA_DATA::qTnp1

New total amounts adsorbed (mol/kg)

Definition at line 94 of file skua.h.

5.67.2.18 double SKUA_DATA::sim_time

Stopping time for the simulation (hrs)

Definition at line 88 of file skua.h.

5.67.2.19 double SKUA_DATA::t

Current time of the simulations (hrs)

Definition at line 90 of file skua.h.

5.67.2.20 double SKUA_DATA::t_counter = 0.0

Counts for print times for output (hrs)

Definition at line 91 of file skua.h.

5.67.2.21 double SKUA_DATA::t_old

Old time of the simulations (hrs)

Definition at line 89 of file skua.h.

5.67.2.22 double SKUA_DATA::t_print

Prints out every t_print time (hrs)

Definition at line 92 of file skua.h.

5.67.2.23 unsigned long int SKUA_DATA::total_steps

Running total of all calculation steps.

Definition at line 86 of file skua.h.

5.67.2.24 const void* SKUA_DATA::user_data

Data structure for user's information needed in parameter functions.

Definition at line 108 of file skua.h.

5.67.2.25 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

5.68 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

5.68.1 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.

5.68.2 Member Data Documentation

5.68.2.1 double SKUA_PARAM::activation_energy

Definition at line 58 of file skua.h.

5.68.2.2 bool SKUA_PARAM::Adsorbable

Definition at line 73 of file skua.h.

5.68.2.3 double SKUA_PARAM::affinity

Definition at line 61 of file skua.h.

5.68.2.4 double SKUA_PARAM::film_transfer

Definition at line 63 of file skua.h.

5.68.2.5 double SKUA_PARAM::Qstn

Definition at line 68 of file skua.h.

5.68.2.6 double SKUA_PARAM::Qstnp1

Definition at line 69 of file skua.h.

5.68.2.7 double SKUA_PARAM::ref_diffusion

Definition at line 59 of file skua.h.

5.68.2.8 double SKUA_PARAM::ref_pressure

Definition at line 62 of file skua.h.

5.68.2.9 double SKUA_PARAM::ref_temperature

Definition at line 60 of file skua.h.

5.68.2.10 std::string SKUA_PARAM::speciesName

Definition at line 75 of file skua.h.

5.68.2.11 double SKUA_PARAM::xIC

Definition at line 65 of file skua.h.

5.68.2.12 double SKUA_PARAM::xn

Definition at line 70 of file skua.h.

5.68.2.13 double SKUA_PARAM::xnp1

Definition at line 71 of file skua.h.

5.68.2.14 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

5.69 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.

5.69.1 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.

5.69.2 Member Data Documentation

5.69.2.1 double SYSTEM_DATA::As

Specific surface area of adsorbent (m^2/kg)

Definition at line 146 of file magpie.h.

5.69.2.2 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.

5.69.2.3 bool SYSTEM_DATA::Carrier

If there is an inert carrier gas, Carrier == true.

Definition at line 155 of file magpie.h.

5.69.2.4 int SYSTEM_DATA::I

Definition at line 148 of file magpie.h.

5.69.2.5 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.

5.69.2.6 int SYSTEM_DATA::J

Definition at line 148 of file magpie.h.

5.69.2.7 int SYSTEM_DATA::K

Special indices used to keep track of sub-systems.

Definition at line 148 of file magpie.h.

5.69.2.8 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.

5.69.2.9 int SYSTEM_DATA::N

Total Number of Components.

Definition at line 147 of file magpie.h.

5.69.2.10 bool SYSTEM_DATA::Output

Boolean to suppress output if desired (true = display, false = no display.

Definition at line 157 of file magpie.h.

5.69.2.11 int SYSTEM_DATA::Par

Number of binary parameters to solve for.

Definition at line 153 of file magpie.h.

5.69.2.12 double SYSTEM_DATA::PI

Total Lumped Spreading Pressure (mol/kg)

Definition at line 144 of file magpie.h.

5.69.2.13 double SYSTEM_DATA::pi

Actual Spreading pressure (J/m^2)

Definition at line 145 of file magpie.h.

5.69.2.14 double SYSTEM_DATA::PT

Total Pressure (kPa)

Definition at line 142 of file magpie.h.

5.69.2.15 double SYSTEM_DATA::qT

Total Amount adsorbed (mol/kg)

Definition at line 143 of file magpie.h.

5.69.2.16 bool SYSTEM_DATA::Recover

If Recover == false, standard GPAST using y's as knowns.

Definition at line 154 of file magpie.h.

5.69.2.17 int SYSTEM_DATA::Sys

Number of sub-systems to solve.

Definition at line 152 of file magpie.h.

5.69.2.18 double SYSTEM_DATA::T

System Temperature (K)

Definition at line 141 of file magpie.h.

5.69.2.19 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:

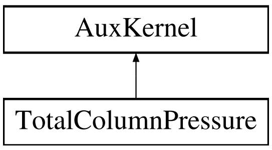
- magpie.h

5.70 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.

5.70.1 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.

5.70.2 Constructor & Destructor Documentation

5.70.2.1 TotalColumnPressure::TotalColumnPressure (const InputParameters & *parameters*)

Standard MOOSE public constructor.

5.70.3 Member Function Documentation

5.70.3.1 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.

5.70.4 Member Data Documentation

5.70.4.1 std::vector<VariableValue *> TotalColumnPressure::_gas_conc [private]

Pointer list for the non-linear concentration variables.

Definition at line 70 of file TotalColumnPressure.h.

5.70.4.2 std::vector<unsigned int> TotalColumnPressure::_index [private]

Indices of the gaseous species coupled to the object.

Definition at line 69 of file TotalColumnPressure.h.

5.70.4.3 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:

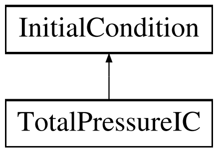
- TotalColumnPressure.h

5.71 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)

5.71.1 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.

5.71.2 Constructor & Destructor Documentation

5.71.2.1 TotalPressureIC::TotalPressureIC (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.71.3 Member Function Documentation

5.71.3.1 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.

5.71.4 Member Data Documentation

5.71.4.1 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:

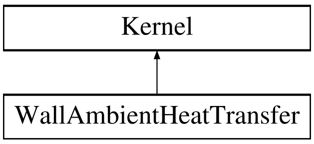
- TotalPressureIC.h

5.72 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.

5.72.1 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.

5.72.2 Constructor & Destructor Documentation

5.72.2.1 WallAmbientHeatTransfer::WallAmbientHeatTransfer (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.72.3 Member Function Documentation

5.72.3.1 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.

5.72.3.2 virtual Real WallAmbientHeatTransfer::computeQpResidual ( ) `[protected]`, `[virtual]`

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.72.4 Member Data Documentation

5.72.4.1 VariableValue& WallAmbientHeatTransfer::_ambient_temp `[private]`

Reference to the outside temperature coupled non-linear variable.

Definition at line 77 of file WallAmbientHeatTransfer.h.

5.72.4.2 const MaterialProperty<Real>& WallAmbientHeatTransfer::_inner_dia `[private]`

Reference to the wall inner diameter material property.

Definition at line 74 of file WallAmbientHeatTransfer.h.

5.72.4.3 const MaterialProperty<Real>& WallAmbientHeatTransfer::_outer_dia `[private]`

Reference to the wall outer diameter material property.

Definition at line 75 of file WallAmbientHeatTransfer.h.

5.72.4.4 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:

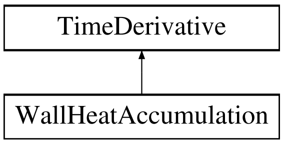
- WallAmbientHeatTransfer.h

5.73 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.

5.73.1 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.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 WallHeatAccumulation::WallHeatAccumulation (const InputParameters & *parameters*)

Required constructor for objects in MOOSE.

5.73.3 Member Function Documentation

5.73.3.1 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.

5.73.3.2 virtual Real WallHeatAccumulation::computeQpResidual ( ) [protected], [virtual]

Required residual function for standard kernels in MOOSE.

This function returns a residual contribution for this object.

5.73.4 Member Data Documentation

5.73.4.1 const MaterialProperty<Real>& WallHeatAccumulation::_wall_density [private]

Reference to the wall density material property.

Definition at line 71 of file WallHeatAccumulation.h.

5.73.4.2 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"
#include "flock.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 Aux_LDF.h File Reference

Generic auxillary kernel to calculate the value of an aux variable using LDF kinetics.

```
#include "AuxKernel.h"
```

Classes

- class Aux_LDF

  Aux_LDF class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< Aux_LDF > ()

6.4.1 Detailed Description

Generic auxillary kernel to calculate the value of an aux variable using LDF kinetics. This file is responsible for calculating the value of the aux variable based on an implicit integration of the linear driving force expression. It's intended use will be to create a generic class that can be inherited by a more specific class to have certain values overriden that may be coulped to other non-linear variables in the simuation. Coupling between this aux kernel and other non-linear variables should be done "loosely" as the intent will ultimately be to couple multi-scale physical phenomena. DO NOT try to fully couple this with non-linear variables. The convergence of the overall system may suffer.

Author

Austin Ladshaw

Date

02/04/2016

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 Aux_LDF.h.

6.4.2 Function Documentation

6.4.2.1 template<> InputParameters validParams< Aux_LDF >( )

6.5 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 > ()

6.5.1 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.

6.5.2 Function Documentation

6.5.2.1 template<> InputParameters validParams< BedHeatAccumulation >( )

6.6 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 > ()

6.6.1 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

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 BedMassAccumulation.h.

6.6.2 Function Documentation

6.6.2.1 template<> InputParameters validParams< BedMassAccumulation >( )

6.7 BedProperties.h File Reference

```
#include "Material.h"
```

Classes

- class BedProperties

  BedProperties class object inherits from Material object.

Functions

- template<>
  InputParameters validParams< BedProperties > ()

6.7.1 Function Documentation

6.7.1.1 template<> InputParameters validParams< BedProperties >( )

6.8 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 > ()

6.8.1 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.

6.8.2 Function Documentation

6.8.2.1 template<> InputParameters validParams< BedWallHeatTransfer >( )

6.9 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 > ()

6.9.1 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.

6.9.2 Function Documentation

6.9.2.1 template<> InputParameters validParams< ColumnTemperatureIC >( )

6.10 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 > ()

6.10.1 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

Definition in file ConcentrationIC.h.

6.10.2 Function Documentation

6.10.2.1 template<> InputParameters validParams< ConcentrationIC >( )

6.11 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 > ()

6.11.1 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.

6.11.2 Macro Definition Documentation

6.11.2.1 #define COUPLEDLDF_H

Definition at line 43 of file CoupledLDF.h.

6.11.3 Function Documentation

6.11.3.1 template<> InputParameters validParams< CoupledLDF >( )

6.12 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 > ()

6.12.1 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.

6.12.2 Function Documentation

6.12.2.1 template<> InputParameters validParams< DGAdvection >( )

6.13 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 > ()

6.13.1 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.

6.13.2 Function Documentation

6.13.2.1 template<> InputParameters validParams< DGAnisotropicDiffusion >( )

6.14 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 > ()

6.14.1 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

Definition in file DGColumnHeatAdvection.h.

6.14.2 Macro Definition Documentation

6.14.2.1 #define DGCOLUMNHEATADVECTION_H

Definition at line 47 of file DGColumnHeatAdvection.h.

6.14.3 Function Documentation

6.14.3.1 template<> InputParameters validParams< DGColumnHeatAdvection >( )

6.15 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 > ()

6.15.1 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

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 DGColumnHeatDispersion.h.

6.15.2 Macro Definition Documentation

6.15.2.1 #define DGCOLUMNHEATDISPERSION_H

Definition at line 47 of file DGColumnHeatDispersion.h.

6.15.3 Function Documentation

6.15.3.1 template<> InputParameters validParams< DGColumnHeatDispersion > ( )

6.16 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 > ()

6.16.1 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.

6.16.2 Function Documentation

6.16.2.1 template<> InputParameters validParams< DGColumnMassAdvection >( )

6.17 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 > ()

6.17.1 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

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 DGColumnMassDispersion.h.

6.17.2 Function Documentation

6.17.2.1 template<> InputParameters validParams< DGColumnMassDispersion > ( )

6.18 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 > ()

6.18.1 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.

6.18.2 Function Documentation

6.18.2.1 template<> InputParameters validParams< DGColumnWallHeatFluxBC >( )

6.19 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 > ()

6.19.1 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.

6.19.2 Function Documentation

6.19.2.1 template<> InputParameters validParams< DGColumnWallHeatFluxLimitedBC > ( )

6.20 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 > ()

6.20.1 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

Definition in file DGFluxBC.h.

6.20.2 Function Documentation

6.20.2.1 template<> InputParameters validParams< DGFluxBC > ( )

6.21 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 > ()

6.21.1 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 penalized until the condition is met.

Author

Austin Ladshaw

Date

11/20/2015

Copyright

Definition in file DGFluxLimitedBC.h.

6.21.2 Function Documentation

6.21.2.1 template<> InputParameters validParams< DGFluxLimitedBC >( )

6.22 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 > ()

6.22.1 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.

6.22.2 Function Documentation

6.22.2.1 template<> InputParameters validParams< DGHeatFluxBC >( )

6.23 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 > ()

6.23.1 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

Definition in file DGHeatFluxLimitedBC.h.

6.23.2 Function Documentation

6.23.2.1 template<> InputParameters validParams< DGHeatFluxLimitedBC >( )

6.24 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 > ()

6.24.1 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

Definition in file DGMassFluxBC.h.

6.24.2 Function Documentation

6.24.2.1 template<> InputParameters validParams< DGMassFluxBC >( )

6.25 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 > ()

6.25.1 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

Definition in file DGMassFluxLimitedBC.h.

6.25.2 Function Documentation

6.25.2.1 template<> InputParameters validParams< DGMassFluxLimitedBC > ( )

6.26 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 > ()

6.26.1 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.

6.26.2 Function Documentation

6.26.2.1 template<> InputParameters validParams< DgospreyApp > ( )

6.27 DgospreyRevision.h File Reference

Macros

- #define DGOSPREY_REVISION "git commit d05ee1e on 2016-02-11"

6.27.1 Macro Definition Documentation

6.27.1.1 #define DGOSPREY_REVISION "git commit d05ee1e on 2016-02-11"

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

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 PSI(*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

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

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 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

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

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 (F-OM). 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 GMRESLP. 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

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 SYSTE-M_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

Definition in file MAGPIE_AdsorptionHeat.h.

6.44.2 Function Documentation

6.44.2.1 template<> InputParameters validParams< **MAGPIE_AdsorptionHeat** > ( )

6.45 MAGPIE_ConstLDF_Adsorption.h File Reference

Auxillary kernel to calculate adsorption based on LDF kinetics with constant coefficients.

```
#include "Aux_LDF.h"
#include "flock.h"
```

Classes

- class MAGPIE_ConstLDF_Adsorption

  MAGPIE_ConstLDF class inherits from AuxKernel.

Macros

- #define MAGPIE_ConstLDF_Adsorption_H

Functions

- template<>
  InputParameters validParams< MAGPIE_ConstLDF_Adsorption > ()

6.45.1 Detailed Description

Auxillary kernel to calculate adsorption based on LDF kinetics with constant coefficients. This file is responsible for calculating the adsorption based on linear driving force kinetics implicitly for the aux variable object. That calculation is based on assuming a constant ldf coefficient, but updates the driving value at every iteration based on a MAGPIE simulation that estimates the new equilibrium point for that aux variable. Remember, it is intended that this kernel will be loosely coupled to the non-linear variables. Otherwise, this gives poor performance and may not converge. We use loose coupling because of the multiscale nature of the physics; we are coupling macro-scale transport to micro-scale equilibria and kinetics.

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

02/04/2016

Copyright

Definition in file MAGPIE_ConstLDF_Adsorption.h.

6.45.2 Macro Definition Documentation

6.45.2.1 #define MAGPIE_ConstLDF_Adsorption_H

Definition at line 51 of file MAGPIE_ConstLDF_Adsorption.h.

6.45.3 Function Documentation

6.45.3.1 template<> InputParameters validParams< MAGPIE_ConstLDF_Adsorption > ( )

6.46 MAGPIE_ConstLDF_Perturbation.h File Reference

Auxillary kernel to calculate adsorption perturbation based on LDF kinetics with constant coefficients.

```
#include "Aux_LDF.h"
#include "flock.h"
```

Classes

- class MAGPIE_ConstLDF_Perturbation
  MAGPIE_ConstLDF class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< MAGPIE_ConstLDF_Perturbation > ()

6.46.1 Detailed Description

Auxillary kernel to calculate adsorption perturbation based on LDF kinetics with constant coefficients. This file is responsible for calculating the adsorption perturbation based on linear driving force kinetics implicitly for the aux variable object. That calculation is based on assuming a constant ldf coefficient, but updates the driving value at every iteration based on a MAGPIE simulation that estimates the new equilibrium point for that aux variable. Remember, it is intended that this kernel will be loosely coupled to the non-linear variables. Otherwise, this gives poor performance and may not converge. We use loose coupling because of the multiscale nature of the physics; we are coupling macro-scale transport to micro-scale equilibria and kinetics.

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

02/04/2016

Copyright

Definition in file MAGPIE_ConstLDF_Perturbation.h.

6.46.2 Function Documentation

6.46.2.1 template<> InputParameters validParams< **MAGPIE_ConstLDF_Perturbation** > ( )

6.47 MAGPIE_MaterialLDF_Adsorption.h File Reference

Auxillary kernel to calculate adsorption based on LDF kinetics with material property coefficients.

```
#include "Aux_LDF.h"
#include "flock.h"
```

Classes

- class MAGPIE_MaterialLDF_Adsorption
  MAGPIE_MaterialLDF_Adsorption class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< MAGPIE_MaterialLDF_Adsorption > ()

6.47.1 Detailed Description

Auxillary kernel to calculate adsorption based on LDF kinetics with material property coefficients. This file is responsible for calculating the adsorption based on linear driving force kinetics implicitly for the aux variable object. That calculation is based on material properties to estimate the ldf coefficient, and updates the driving value at every iteration based on a MAGPIE simulation that estimates the new equilibrium point for that aux variable. Remember, it is intended that this kernel will be loosely coupled to the non-linear variables. Otherwise, this gives poor performance and may not converge. We use loose coupling because of the multiscale nature of the physics; we are coupling macro-scale transport to micro-scale equilibria and kinetics.

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

02/05/2016

Copyright

Definition in file MAGPIE_MaterialLDF_Adsorption.h.

6.47.2 Function Documentation

6.47.2.1 template<> InputParameters validParams< MAGPIE_MaterialLDF_Adsorption > ( )

6.48 MAGPIE_MaterialLDF_Perturbation.h File Reference

Auxillary kernel to calculate adsorption perturbation based on LDF kinetics with material property coefficients.

```
#include "Aux_LDF.h"
#include "flock.h"
```

Classes

- class MAGPIE_MaterialLDF_Perturbation
  MAGPIE_MaterialLDF_Perturbation class inherits from AuxKernel.

Functions

- template<>
  InputParameters validParams< MAGPIE_MaterialLDF_Perturbation > ()

6.48.1 Detailed Description

Auxillary kernel to calculate adsorption perturbation based on LDF kinetics with material property coefficients. This file is responsible for calculating the adsorption perturbation based on linear driving force kinetics implicitly for the aux variable object. That calculation is based on material properties to estimate the ldf coefficient, and updates the driving value at every iteration based on a MAGPIE simulation that estimates the new equilibrium point for that aux variable. Remember, it is intended that this kernel will be loosely coupled to the non-linear variables. Otherwise, this gives poor performance and may not converge. We use loose coupling because of the multiscale nature of the physics; we are coupling macro-scale transport to micro-scale equilibria and kinetics.

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

02/05/2016

Copyright

Definition in file MAGPIE_MaterialLDF_Perturbation.h.

6.48.2 Function Documentation

6.48.2.1 template<> InputParameters validParams< MAGPIE_MaterialLDF_Perturbation >( )

6.49 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 > ()

6.49.1 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

Definition in file MAGPIE_Perturbation.h.

6.49.2 Function Documentation

6.49.2.1 template<> InputParameters validParams< **MAGPIE_Perturbation** >( )

6.50 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 > ()

6.50.1 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

Definition in file MagpieAdsorbateProperties.h.

6.50.2 Function Documentation

6.50.2.1 template<> InputParameters validParams< **MagpieAdsorbateProperties** >( )

6.51 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.

6.51.1 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.

6.51.2 Macro Definition Documentation

6.51.2.1 #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.

6.51.2.2 #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.

6.51.2.3 #define Dp(*Dm,* *ep*) (ep∗ep∗Dm)

Estimate of Pore Diffusivity (cm^2/s)

Definition at line 29 of file scopsowl.h.

6.51.2.4 #define SCOPSOWL_HPP_

Definition at line 26 of file scopsowl.h.

6.51.3 Function Documentation

6.51.3.1 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

6.51.3.2 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

6.51.3.3 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

6.51.3.4 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

6.51.3.5 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

6.51.3.6 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

6.51.3.7 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

6.51.3.8 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

6.51.3.9 void print2file_SCOPSOWL_header (SCOPSOWL_DATA ∗ *owl_dat*)

Function to call the species and time header functions.

6.51.3.10 void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA ∗ *owl_dat*)

Function to print out the new time results to the output file.

6.51.3.11 void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA ∗ *owl_dat*)

Function to print out the old time results to the output file.

6.51.3.12 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.

6.51.3.13 void print2file_species_header (FILE ∗ *Output*, SCOPSOWL_DATA ∗ *owl_dat*, int *i*)

Function to print out the main header for the output file.

6.51.3.14 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)

6.51.3.15 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)

6.51.3.16 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)

6.51.3.17 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)

6.51.3.18 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)

6.51.3.19 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)

6.51.3.20 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)

6.51.3.21 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)

6.51.3.22 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

6.52 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.

6.52.1 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.

6.52.2 Macro Definition Documentation

6.52.2.1 #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.

6.52.2.2 #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.

6.52.2.3 #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.

6.52.2.4 #define SKUA_HPP_

Definition at line 37 of file skua.h.

6.52.3 Function Documentation

6.52.3.1 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

6.52.3.2 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

6.52.3.3 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

6.52.3.4 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

6.52.3.5 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

6.52.3.6 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.

6.52.3.7 void print2file_SKUA_header (SKUA_DATA * *skua_dat*)

Function calls the other header functions to establish output file structure.

6.52.3.8 void print2file_SKUA_results_new (SKUA_DATA * *skua_dat*)

Function to print out the new time step simulation results to the output file.

6.52.3.9 void print2file_SKUA_results_old (SKUA_DATA * *skua_dat*)

Function to print out the old time step simulation results to the output file.

6.52.3.10 void print2file_SKUA_time_header (FILE * *Output*, SKUA_DATA * *skua_dat*, int *i*)

Function to print out time and space headers to output file.

6.52.3.11 void print2file_species_header (FILE * *Output*, SKUA_DATA * *skua_dat*, int *i*)

Function to print out the species' headers to output file.

6.52.3.12 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.

6.52.3.13 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.

6.52.3.14 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.

6.52.3.15 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

6.52.3.16 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

6.52.3.17 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.

6.52.3.18 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).

6.52.3.19 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.

6.52.3.20 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).

6.52.3.21 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.

6.52.3.22 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

6.53 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 > ()

6.53.1 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

Definition in file TotalColumnPressure.h.

6.53.2 Function Documentation

6.53.2.1 template<> InputParameters validParams< **TotalColumnPressure** > ( )

6.54 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 > ()

6.54.1 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 TotalPressureIC.h.

6.54.2 Function Documentation

6.54.2.1 template<> InputParameters validParams< **TotalPressureIC** > ( )

6.55 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 > ()

6.55.1 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

Definition in file WallAmbientHeatTransfer.h.

6.55.2 Function Documentation

6.55.2.1 template<> InputParameters validParams< WallAmbientHeatTransfer >( )

6.56 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 > ()

6.56.1 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

Definition in file WallHeatAccumulation.h.

6.56.2 Function Documentation

6.56.2.1 template<> InputParameters validParams< WallHeatAccumulation >( )

Index

~DgospreyApp
DgospreyApp, 86
~Matrix
Matrix, 161
_Diffusion
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
DGFluxBC, 66
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 79
DGMassFluxLimitedBC, 83
GAnisotropicDiffusion, 109
GColumnHeatDispersion, 114
GColumnMassDispersion, 118
_Dxx
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
DGFluxBC, 66
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 79
DGMassFluxLimitedBC, 83
GAnisotropicDiffusion, 109
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dxy
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
DGFluxBC, 66
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 83
GAnisotropicDiffusion, 109
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dxz
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 66
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 83
GAnisotropicDiffusion, 109
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dyx
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 83
GAnisotropicDiffusion, 109
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dyy
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 69
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
GAnisotropicDiffusion, 110
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dyz
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 52
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 69
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
GAnisotropicDiffusion, 110
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dzx

DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 53
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 69
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
GAnisotropicDiffusion, 110
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dzy
DGAnisotropicDiffusion, 47
DGColumnHeatDispersion, 53
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 69
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
GAnisotropicDiffusion, 110
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Dzz
DGAnisotropicDiffusion, 48
DGColumnHeatDispersion, 53
DGColumnMassDispersion, 58
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 70
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 76
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
GAnisotropicDiffusion, 110
GColumnHeatDispersion, 114
GColumnMassDispersion, 119
_Kz
BedProperties, 28
_PT_IC
ConcentrationIC, 41
TotalPressureIC, 205
_TC_IC
ColumnTemperatureIC, 39
_T_IC
ConcentrationIC, 41
_Ua
BedProperties, 29
_Uw
BedProperties, 29
_act_energy
AdsorbentProperties, 12
_affinity
AdsorbentProperties, 12
_ambient_temp
WallAmbientHeatTransfer, 207
_bed_wall_transfer_coeff
BedProperties, 27
BedWallHeatTransfer, 31
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
_binder_fraction
AdsorbentProperties, 12
_binder_porosity
AdsorbentProperties, 13
FlowProperties, 103
MAGPIE_MaterialLDF_Adsorption, 148
MAGPIE_MaterialLDF_Perturbation, 151
_binder_ratio
AdsorbentProperties, 13
_coef
CoupledLDF, 43
LinearDrivingForce, 139
_column_length
FlowProperties, 103
_column_temp
BedWallHeatTransfer, 31
_comp_Sutherland_const
FlowProperties, 103
_comp_heat_capacity
FlowProperties, 103
_comp_ref_temp
FlowProperties, 103
_comp_ref_viscosity
FlowProperties, 103
_conductivity
BedProperties, 27
DGColumnHeatDispersion, 52
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 76
GColumnHeatDispersion, 114
_crystal_rad
AdsorbentProperties, 13
_crystal_radius
AdsorbentProperties, 13
MAGPIE_MaterialLDF_Adsorption, 148
MAGPIE_MaterialLDF_Perturbation, 151
_din
BedProperties, 27
_dispersion
DGColumnMassDispersion, 57
DGMassFluxBC, 79
DGMassFluxLimitedBC, 83
FlowProperties, 103
GColumnMassDispersion, 118
_dout
BedProperties, 28

_drive_coef
CoupledLDF, 43
_drive_var
CoupledLDF, 43
_driving_value
Aux_LDF, 21
CoupledLDF, 43
LinearDrivingForce, 139
MAGPIE_ConstLDF_Adsorption, 144
MAGPIE_ConstLDF_Perturbation, 145
MAGPIE_MaterialLDF_Adsorption, 148
MAGPIE_MaterialLDF_Perturbation, 151
_eb
BedProperties, 28
_enthalpy_1
MagpieAdsorbateProperties, 156
_enthalpy_2
MagpieAdsorbateProperties, 156
_enthalpy_3
MagpieAdsorbateProperties, 156
_enthalpy_4
MagpieAdsorbateProperties, 156
_enthalpy_5
MagpieAdsorbateProperties, 156
_enthalpy_6
MagpieAdsorbateProperties, 156
_entropy_1
MagpieAdsorbateProperties, 156
_entropy_2
MagpieAdsorbateProperties, 156
_entropy_3
MagpieAdsorbateProperties, 156
_entropy_4
MagpieAdsorbateProperties, 156
_entropy_5
MagpieAdsorbateProperties, 157
_entropy_6
MagpieAdsorbateProperties, 157
_eps_binder
AdsorbentProperties, 13
_epsilon
DGAnisotropicDiffusion, 48
DGColumnHeatDispersion, 53
DGColumnMassDispersion, 58
DGColumnWallHeatFluxLimitedBC, 64
DGFluxLimitedBC, 70
DGHeatFluxLimitedBC, 76
DGMassFluxLimitedBC, 84
_film_transfer
FlowProperties, 103
MAGPIE_MaterialLDF_Adsorption, 148
MAGPIE_MaterialLDF_Perturbation, 151
_flow_rate
FlowProperties, 103
_gaining
CoupledLDF, 43
LinearDrivingForce, 139
_gas_conc
AdsorbentProperties, 13
BedProperties, 28
FlowProperties, 103
MagpieAdsorbateProperties, 157
TotalColumnPressure, 204
_gas_conc_old
MagpieAdsorbateProperties, 157
_gas_const
FlowProperties.h, 249
_gas_density
DGColumnHeatAdvection, 50
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 76
FlowProperties, 104
GColumnHeatAdvection, 112
_gas_heat_capacity
DGColumnHeatAdvection, 50
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
FlowProperties, 104
GColumnHeatAdvection, 112
_gas_molecular_wieght
FlowProperties, 104
_gas_viscosity
FlowProperties, 104
_heat_retardation
BedHeatAccumulation, 24
FlowProperties, 104
_hs
AdsorbentProperties, 13
_hw
BedProperties, 28
_index
AdsorbentProperties, 13
BedMassAccumulation, 25
BedProperties, 28
DGColumnMassDispersion, 58
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
FlowProperties, 104
GColumnMassDispersion, 119
MAGPIE_Adsorption, 141
MAGPIE_AdsorptionHeat, 142
MAGPIE_ConstLDF_Adsorption, 144
MAGPIE_ConstLDF_Perturbation, 145
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 151
MAGPIE_Perturbation, 153
MagpieAdsorbateProperties, 157
TotalColumnPressure, 204
_inner_dia
BedProperties, 28
BedWallHeatTransfer, 31
FlowProperties, 104
WallAmbientHeatTransfer, 207
_input_molefraction
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84

_input_pressure
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
_input_temperature
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
_ldf_coef
Aux_LDF, 21
MAGPIE_ConstLDF_Adsorption, 144
MAGPIE_ConstLDF_Perturbation, 146
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 151
_length
BedProperties, 28
_macropore_radius
AdsorbentProperties, 13
_magpie_dat
AdsorbentProperties, 13
MAGPIE_Adsorption, 141
MAGPIE_AdsorptionHeat, 142
MAGPIE_ConstLDF_Adsorption, 144
MAGPIE_ConstLDF_Perturbation, 146
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 152
MAGPIE_Perturbation, 153
MagpieAdsorbateProperties, 157
_magpie_dat_old
MagpieAdsorbateProperties, 157
_max_capacity
MagpieAdsorbateProperties, 157
_mixed_gas
FlowProperties, 104
_mixed_gas_old
FlowProperties, 104
_molar_volume
MagpieAdsorbateProperties, 157
_molecular_diffusion
DGColumnMassDispersion, 58
DGMassFluxBC, 80
DGMassFluxLimitedBC, 84
FlowProperties, 104
GColumnMassDispersion, 119
_molecular_weight
FlowProperties, 104
_num_sites
MagpieAdsorbateProperties, 157
_outer_dia
BedProperties, 28
BedWallHeatTransfer, 31
WallAmbientHeatTransfer, 207
_pellet_density
AdsorbentProperties, 13
AdsorptionHeatAccumulation, 16
AdsorptionMassTransfer, 18
FlowProperties, 105
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 152
_pellet_dia
AdsorbentProperties, 14
_pellet_diameter
AdsorbentProperties, 14
FlowProperties, 105
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 152
_pellet_heat_capacity
AdsorbentProperties, 14
FlowProperties, 105
_pore_diffusion
FlowProperties, 105
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 152
_pore_size
AdsorbentProperties, 14
FlowProperties, 105
_porosity
AdsorptionHeatAccumulation, 16
AdsorptionMassTransfer, 18
BedProperties, 28
FlowProperties, 105
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 152
_ref_diff
AdsorbentProperties, 14
_ref_temp
AdsorbentProperties, 14
_retardation
BedMassAccumulation, 25
FlowProperties, 105
_rhos
AdsorbentProperties, 14
_rhow
BedProperties, 28
_sigma
DGAnisotropicDiffusion, 48
DGColumnHeatDispersion, 53
DGColumnMassDispersion, 58
DGColumnWallHeatFluxLimitedBC, 64
DGFluxLimitedBC, 70
DGHeatFluxLimitedBC, 77
DGMassFluxLimitedBC, 84
_solid
AdsorptionMassTransfer, 18
_solid_conc
FlowProperties, 105
MAGPIE_AdsorptionHeat, 142
_solid_heat
AdsorptionHeatAccumulation, 16
_solid_heat_old
AdsorptionHeatAccumulation, 16
_solid_old
AdsorptionMassTransfer, 18
_solid_perturb
FlowProperties, 105
_surface_diffusion

AdsorbentProperties, 14
MAGPIE_MaterialLDF_Adsorption, 149
MAGPIE_MaterialLDF_Perturbation, 152
_temperature
AdsorbentProperties, 14
BedProperties, 29
FlowProperties, 105
MagpieAdsorbateProperties, 158
TotalColumnPressure, 204
_total_pressure
FlowProperties, 105
MagpieAdsorbateProperties, 158
_u_input
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 70
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 81
DGMassFluxLimitedBC, 84
_var
CoupledLDF, 43
LinearDrivingForce, 139
_vel
DGColumnHeatAdvection, 50
DGColumnMassAdvection, 55
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 81
DGMassFluxLimitedBC, 85
GColumnHeatAdvection, 112
GColumnMassAdvection, 116
_velocity
DGAdvection, 45
DGColumnHeatAdvection, 50
DGColumnMassAdvection, 55
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 70
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 81
DGMassFluxLimitedBC, 85
FlowProperties, 106
GAdvection, 107
GColumnHeatAdvection, 112
GColumnMassAdvection, 116
_vx
DGAdvection, 45
DGColumnHeatAdvection, 50
DGColumnMassAdvection, 55
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 70
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 81
DGMassFluxLimitedBC, 85
GAdvection, 107
GColumnHeatAdvection, 112
GColumnMassAdvection, 116
_vy
DGAdvection, 45
DGColumnHeatAdvection, 50
DGColumnMassAdvection, 55
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 70
DGHeatFluxBC, 73
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 81
DGMassFluxLimitedBC, 85
GAdvection, 107
GColumnHeatAdvection, 112
GColumnMassAdvection, 116
_vz
DGAdvection, 45
DGColumnHeatAdvection, 50
DGColumnMassAdvection, 55
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 64
DGFluxBC, 67
DGFluxLimitedBC, 70
DGHeatFluxBC, 74
DGHeatFluxLimitedBC, 77
DGMassFluxBC, 81
DGMassFluxLimitedBC, 85
GAdvection, 107
GColumnHeatAdvection, 112
GColumnMassAdvection, 116
_wall_density
BedProperties, 29
WallHeatAccumulation, 209
_wall_exterior_transfer_coeff
BedProperties, 29
WallAmbientHeatTransfer, 207
_wall_heat_capacity
BedProperties, 29
WallHeatAccumulation, 209
_wall_temp
DGColumnWallHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 65
_y_IC
ConcentrationIC, 41

A
magpie.h, 273
ARNOLDI_DATA, 18
beta, 19
e1, 19
Hkp1, 19
hp1, 19
iter, 19

k, 19
Output, 19
sum, 19
v, 20
Vk, 20
w, 20
yk, 20
activation_energy
SCOPSOWL_PARAM_DATA, 193
SKUA_PARAM, 199
adjoint
Matrix, 161
Adsorbable
SCOPSOWL_PARAM_DATA, 193
SKUA_PARAM, 199
AdsorbentProperties, 10
_act_energy, 12
_affinity, 12
_binder_fraction, 12
_binder_porosity, 13
_binder_ratio, 13
_crystal_rad, 13
_crystal_radius, 13
_eps_binder, 13
_gas_conc, 13
_hs, 13
_index, 13
_macropore_radius, 13
_magpie_dat, 13
_pellet_density, 13
_pellet_dia, 14
_pellet_diameter, 14
_pellet_heat_capacity, 14
_pore_size, 14
_ref_diff, 14
_ref_temp, 14
_rhos, 14
_surface_diffusion, 14
_temperature, 14
AdsorbentProperties, 12
AdsorbentProperties, 12
computeQpProperties, 12
AdsorbentProperties.h, 209
validParams< AdsorbentProperties >, 210
AdsorptionHeatAccumulation, 15
_pellet_density, 16
_porosity, 16
_solid_heat, 16
_solid_heat_old, 16
AdsorptionHeatAccumulation, 15
AdsorptionHeatAccumulation, 15
computeQpJacobian, 16
computeQpResidual, 16
AdsorptionHeatAccumulation.h, 210
validParams< AdsorptionHeatAccumulation >, 211
AdsorptionMassTransfer, 16
_pellet_density, 18
_porosity, 18
_solid, 18
_solid_old, 18
AdsorptionMassTransfer, 17
AdsorptionMassTransfer, 17
computeQpJacobian, 17
computeQpResidual, 17
AdsorptionMassTransfer.h, 211
validParams< AdsorptionMassTransfer >, 212
affinity
SCOPSOWL_PARAM_DATA, 193
SKUA_PARAM, 199
Ai
OPTRANS_DATA, 174
alpha
BACKTRACK_DATA, 22
BiCGSTAB_DATA, 32
CGS_DATA, 36
GCR_DATA, 121
PCG_DATA, 175
anchor_alias_dne
error.h, 240
Ap
PCG_DATA, 175
arg
GMRESR_DATA, 126
arg_matrix_same
error.h, 240
arnoldi
lark.h, 258
arnoldi_dat
GMRESLP_DATA, 124
As
SYSTEM_DATA, 201
associateSyntax
DgospreyApp, 86
Aux_LDF, 20
_driving_value, 21
_ldf_coef, 21
Aux_LDF, 21
Aux_LDF, 21
computeValue, 21
Aux_LDF.h, 212
validParams< Aux_LDF >, 213
avg_norm
SYSTEM_DATA, 201
avgDp
scopsowl.h, 288

BACKTRACK_DATA, 21
alpha, 22
constRho, 22
Fk, 22
fun_call, 22
lambdaMin, 22
normFkp1, 22
rho, 22
xk, 23
backtrack_dat

PJFNK_DATA, 181
backtrackLineSearch
lark.h, 259
BedHeatAccumulation, 23
_heat_retardation, 24
BedHeatAccumulation, 24
BedHeatAccumulation, 24
computeQpJacobian, 24
computeQpResidual, 24
BedHeatAccumulation.h, 213
validParams< BedHeatAccumulation >, 214
BedMassAccumulation, 24
_index, 25
_retardation, 25
BedMassAccumulation, 25
BedMassAccumulation, 25
computeQpJacobian, 25
computeQpResidual, 25
BedMassAccumulation.h, 214
validParams< BedMassAccumulation >, 215
BedProperties, 26
_Kz, 28
_Ua, 29
_Uw, 29
_bed_wall_transfer_coeff, 27
_conductivity, 27
_din, 27
_dout, 28
_eb, 28
_gas_conc, 28
_hw, 28
_index, 28
_inner_dia, 28
_length, 28
_outer_dia, 28
_porosity, 28
_rhow, 28
_temperature, 29
_wall_density, 29
_wall_exterior_transfer_coeff, 29
_wall_heat_capacity, 29
BedProperties, 27
BedProperties, 27
computeQpProperties, 27
BedProperties.h, 215
validParams< BedProperties >, 215
BedWallHeatTransfer, 29
_bed_wall_transfer_coeff, 31
_column_temp, 31
_inner_dia, 31
_outer_dia, 31
BedWallHeatTransfer, 30
BedWallHeatTransfer, 30
computeQpJacobian, 30
computeQpResidual, 30
BedWallHeatTransfer.h, 215
validParams< BedWallHeatTransfer >, 216
bestres
BiCGSTAB_DATA, 32
CGS_DATA, 36
GCR_DATA, 121
GMRESLP_DATA, 124
GMRESRP_DATA, 130
PCG_DATA, 175
PICARD_DATA, 178
bestx
BiCGSTAB_DATA, 33
CGS_DATA, 36
GCR_DATA, 121
GMRESLP_DATA, 124
GMRESRP_DATA, 130
PCG_DATA, 176
PICARD_DATA, 178
PJFNK_DATA, 181
beta
ARNOLDI_DATA, 19
BiCGSTAB_DATA, 33
CGS_DATA, 36
FINCH_DATA, 91
GCR_DATA, 121
PCG_DATA, 176
BiCGSTAB
lark.h, 258
BiCGSTAB_DATA, 31
alpha, 32
bestres, 32
bestx, 33
beta, 33
breakdown, 33
iter, 33
maxit, 33
omega, 33
omega_old, 33
Output, 33
p, 33
r, 33
r0, 33
relres, 34
relres_base, 34
res, 34
rho, 34
rho_old, 34
s, 34
t, 34
tol_abs, 34
tol_rel, 34
v, 34
x, 34
y, 35
z, 35
bicgstab
lark.h, 259
bicgstab_dat
PJFNK_DATA, 181
binary_diffusion
MIXED_GAS, 170

binder_fraction
SCOPSOWL_DATA, 188
binder_poresize
SCOPSOWL_DATA, 188
binder_porosity
SCOPSOWL_DATA, 188
Bounce
PJFNK_DATA, 181
breakdown
BiCGSTAB_DATA, 33
CGS_DATA, 36
GCR_DATA, 121

c
CGS_DATA, 37
GCR_DATA, 121
CGS
lark.h, 258
c_temp
GCR_DATA, 121
CC_E
FINCH_DATA, 91
CC_I
FINCH_DATA, 91
CE3
egret.h, 235
CGS_DATA, 35
alpha, 36
bestres, 36
bestx, 36
beta, 36
breakdown, 36
c, 37
iter, 37
maxit, 37
Output, 37
p, 37
r, 37
r0, 37
relres, 37
relres_base, 37
res, 37
rho, 37
sigma, 38
tol_abs, 38
tol_rel, 38
u, 38
v, 38
w, 38
x, 38
z, 38
CL_E
FINCH_DATA, 91
CL_I
FINCH_DATA, 91
CN
FINCH_DATA, 91
COUPLEDLDF_H
CoupledLDF.h, 219
CR_E
FINCH_DATA, 92
CR_I
FINCH_DATA, 92
calculate_properties
egret.h, 237
callroutine
FINCH_DATA, 91
Carrier
SYSTEM_DATA, 201
Cartesian
finch.h, 243
cgs
lark.h, 260
cgs_dat
PJFNK_DATA, 181
char_length
MIXED_GAS, 170
char_macro
SCOPSOWL_DATA, 188
char_measure
SKUA_DATA, 196
char_micro
SCOPSOWL_DATA, 188
check_Mass
finch.h, 244
CheckMass
FINCH_DATA, 91
CheckMolefractions
MIXED_GAS, 170
cofactor
Matrix, 161
columnExtend
Matrix, 161
columnExtract
Matrix, 162
columnProjection
Matrix, 162
columnReplace
Matrix, 162
columnShrink
Matrix, 162
ColumnTemperatureIC, 38
_TC_IC, 39
ColumnTemperatureIC, 39
ColumnTemperatureIC, 39
value, 39
ColumnTemperatureIC.h, 216
validParams$<$ ColumnTemperatureIC $>$, 217
columnVectorFill
Matrix, 162
columns
Matrix, 162
computeQpJacobian
AdsorptionHeatAccumulation, 16
AdsorptionMassTransfer, 17
BedHeatAccumulation, 24
BedMassAccumulation, 25

BedWallHeatTransfer, 30
CoupledLDF, 42
DGAdvection, 45
DGAnisotropicDiffusion, 47
DGColumnHeatAdvection, 49
DGColumnHeatDispersion, 52
DGColumnMassAdvection, 54
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
DGFluxBC, 66
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 75
DGMassFluxBC, 79
DGMassFluxLimitedBC, 83
GAdvection, 107
GAnisotropicDiffusion, 109
GColumnHeatAdvection, 111
GColumnHeatDispersion, 114
GColumnMassAdvection, 116
GColumnMassDispersion, 118
LinearDrivingForce, 139
WallAmbientHeatTransfer, 207
WallHeatAccumulation, 208
computeQpProperties
AdsorbentProperties, 12
BedProperties, 27
FlowProperties, 102
MagpieAdsorbateProperties, 155
computeQpResidual
AdsorptionHeatAccumulation, 16
AdsorptionMassTransfer, 17
BedHeatAccumulation, 24
BedMassAccumulation, 25
BedWallHeatTransfer, 30
CoupledLDF, 42
DGAdvection, 45
DGAnisotropicDiffusion, 47
DGColumnHeatAdvection, 49
DGColumnHeatDispersion, 52
DGColumnMassAdvection, 54
DGColumnMassDispersion, 57
DGColumnWallHeatFluxBC, 60
DGColumnWallHeatFluxLimitedBC, 63
DGFluxBC, 66
DGFluxLimitedBC, 69
DGHeatFluxBC, 72
DGHeatFluxLimitedBC, 75
DGMassFluxBC, 79
DGMassFluxLimitedBC, 83
GAdvection, 107
GAnisotropicDiffusion, 109
GColumnHeatAdvection, 111
GColumnHeatDispersion, 114
GColumnMassAdvection, 116
GColumnMassDispersion, 118
LinearDrivingForce, 139
WallAmbientHeatTransfer, 207
WallHeatAccumulation, 208
computeValue
Aux_LDF, 21
MAGPIE_Adsorption, 140
MAGPIE_AdsorptionHeat, 142
MAGPIE_ConstLDF_Adsorption, 144
MAGPIE_ConstLDF_Perturbation, 145
MAGPIE_MaterialLDF_Adsorption, 148
MAGPIE_MaterialLDF_Perturbation, 151
MAGPIE_Perturbation, 153
TotalColumnPressure, 204
ConcentrationIC, 40
_PT_IC, 41
_T_IC, 41
_y_IC, 41
ConcentrationIC, 40
ConcentrationIC, 40
value, 41
ConcentrationIC.h, 217
validParams< ConcentrationIC >, 218
const_Dc
skua.h, 295
const_filmMassTransfer
scopsowl.h, 289
const_kf
skua.h, 296
const_pore_diffusion
scopsowl.h, 289
constRho
BACKTRACK_DATA, 22
ConstantICFill
Matrix, 162
coord
SKUA_DATA, 196
coord_macro
SCOPSOWL_DATA, 188
coord_micro
SCOPSOWL_DATA, 188
CoupledLDF, 41
_coef, 43
_drive_coef, 43
_drive_var, 43
_driving_value, 43
_gaining, 43
_var, 43
computeQpJacobian, 42
computeQpResidual, 42
CoupledLDF, 42
CoupledLDF, 42
CoupledLDF.h, 218
COUPLEDLDF_H, 219
validParams< CoupledLDF >, 219
crystal_radius
SCOPSOWL_DATA, 188
Cstd
egret.h, 235
Cylindrical

finch.h, 243
d
FINCH_DATA, 92
D_c
skua.h, 295
D_ii
egret.h, 235
D_ij
egret.h, 236
D_inf
skua.h, 295
D_o
skua.h, 295
DBL_EPSILON
magpie.h, 273
DGAdvection, 43
_velocity, 45
_vx, 45
_vy, 45
_vz, 45
computeQpJacobian, 45
computeQpResidual, 45
DGAdvection, 44
DGAdvection, 44
DGAdvection.h, 219
validParams< DGAdvection >, 220
DGAnisotropicDiffusion, 45
_Diffusion, 47
_Dxx, 47
_Dxy, 47
_Dxz, 47
_Dyx, 47
_Dyy, 47
_Dyz, 47
_Dzx, 47
_Dzy, 47
_Dzz, 48
_epsilon, 48
_sigma, 48
computeQpJacobian, 47
computeQpResidual, 47
DGAnisotropicDiffusion, 47
DGAnisotropicDiffusion, 47
DGAnisotropicDiffusion.h, 220
validParams< DGAnisotropicDiffusion >, 221
DGColumnHeatAdvection, 48
_gas_density, 50
_gas_heat_capacity, 50
_vel, 50
_velocity, 50
_vx, 50
_vy, 50
_vz, 50
computeQpJacobian, 49
computeQpResidual, 49
DGColumnHeatAdvection, 49
DGColumnHeatAdvection, 49
DGColumnHeatAdvection.h, 221
validParams< DGColumnHeatAdvection >, 222
DGColumnHeatDispersion, 50
_Diffusion, 52
_Dxx, 52
_Dxy, 52
_Dxz, 52
_Dyx, 52
_Dyy, 52
_Dyz, 52
_Dzx, 53
_Dzy, 53
_Dzz, 53
_conductivity, 52
_epsilon, 53
_sigma, 53
computeQpJacobian, 52
computeQpResidual, 52
DGColumnHeatDispersion, 52
DGColumnHeatDispersion, 52
DGColumnHeatDispersion.h, 222
validParams< DGColumnHeatDispersion >, 223
DGColumnMassAdvection, 53
_vel, 55
_velocity, 55
_vx, 55
_vy, 55
_vz, 55
computeQpJacobian, 54
computeQpResidual, 54
DGColumnMassAdvection, 54
DGColumnMassAdvection, 54
DGColumnMassAdvection.h, 223
validParams< DGColumnMassAdvection >, 224
DGColumnMassDispersion, 55
_Diffusion, 57
_Dxx, 57
_Dxy, 57
_Dxz, 57
_Dyx, 57
_Dyy, 57
_Dyz, 57
_Dzx, 57
_Dzy, 57
_Dzz, 58
_dispersion, 57
_epsilon, 58
_index, 58
_molecular_diffusion, 58
_sigma, 58
computeQpJacobian, 57
computeQpResidual, 57
DGColumnMassDispersion, 56
DGColumnMassDispersion, 56
DGColumnMassDispersion.h, 224
validParams< DGColumnMassDispersion >, 225
DGColumnWallHeatFluxBC, 58
_Diffusion, 60
_Dxx, 60

_Dxy, 60
_Dxz, 60
_Dyx, 60
_Dyy, 60
_Dyz, 60
_Dzx, 60
_Dzy, 61
_Dzz, 61
_bed_wall_transfer_coeff, 60
_conductivity, 60
_u_input, 61
_velocity, 61
_vx, 61
_vy, 61
_vz, 61
_wall_temp, 61
computeQpJacobian, 60
computeQpResidual, 60
DGColumnWallHeatFluxBC, 59
DGColumnWallHeatFluxBC, 59
DGColumnWallHeatFluxBC.h, 225
validParams< DGColumnWallHeatFluxBC >, 226
DGColumnWallHeatFluxLimitedBC, 61
_Diffusion, 63
_Dxx, 63
_Dxy, 63
_Dxz, 64
_Dyx, 64
_Dyy, 64
_Dyz, 64
_Dzx, 64
_Dzy, 64
_Dzz, 64
_bed_wall_transfer_coeff, 63
_conductivity, 63
_epsilon, 64
_sigma, 64
_u_input, 64
_velocity, 64
_vx, 64
_vy, 64
_vz, 64
_wall_temp, 65
computeQpJacobian, 63
computeQpResidual, 63
DGColumnWallHeatFluxLimitedBC, 63
DGColumnWallHeatFluxLimitedBC, 63
DGColumnWallHeatFluxLimitedBC.h, 226
validParams< DGColumnWallHeatFluxLimitedBC >, 227
DGFluxBC, 65
_Diffusion, 66
_Dxx, 66
_Dxy, 66
_Dxz, 66
_Dyx, 67
_Dyy, 67
_Dyz, 67
_Dzx, 67
_Dzy, 67
_Dzz, 67
_u_input, 67
_velocity, 67
_vx, 67
_vy, 67
_vz, 67
computeQpJacobian, 66
computeQpResidual, 66
DGFluxBC, 66
DGFluxBC, 66
DGFluxBC.h, 227
validParams< DGFluxBC >, 228
DGFluxLimitedBC, 67
_Diffusion, 69
_Dxx, 69
_Dxy, 69
_Dxz, 69
_Dyx, 69
_Dyy, 69
_Dyz, 69
_Dzx, 69
_Dzy, 69
_Dzz, 70
_epsilon, 70
_sigma, 70
_u_input, 70
_velocity, 70
_vx, 70
_vy, 70
_vz, 70
computeQpJacobian, 69
computeQpResidual, 69
DGFluxLimitedBC, 69
DGFluxLimitedBC, 69
DGFluxLimitedBC.h, 228
validParams< DGFluxLimitedBC >, 229
DGHeatFluxBC, 70
_Diffusion, 72
_Dxx, 72
_Dxy, 72
_Dxz, 72
_Dyx, 72
_Dyy, 73
_Dyz, 73
_Dzx, 73
_Dzy, 73
_Dzz, 73
_conductivity, 72
_gas_density, 73
_gas_heat_capacity, 73
_input_temperature, 73
_u_input, 73
_vel, 73
_velocity, 73
_vx, 73
_vy, 73

_vz, 74
computeQpJacobian, 72
computeQpResidual, 72
DGHeatFluxBC, 72
DGHeatFluxBC, 72
DGHeatFluxBC.h, 229
validParams< DGHeatFluxBC >, 230
DGHeatFluxLimitedBC, 74
_Diffusion, 76
_Dxx, 76
_Dxy, 76
_Dxz, 76
_Dyx, 76
_Dyy, 76
_Dyz, 76
_Dzx, 76
_Dzy, 76
_Dzz, 76
_conductivity, 76
_epsilon, 76
_gas_density, 76
_gas_heat_capacity, 77
_input_temperature, 77
_sigma, 77
_u_input, 77
_vel, 77
_velocity, 77
_vx, 77
_vy, 77
_vz, 77
computeQpJacobian, 75
computeQpResidual, 75
DGHeatFluxLimitedBC, 75
DGHeatFluxLimitedBC, 75
DGHeatFluxLimitedBC.h, 230
validParams< DGHeatFluxLimitedBC >, 231
DGMassFluxBC, 77
_Diffusion, 79
_Dxx, 79
_Dxy, 80
_Dxz, 80
_Dyx, 80
_Dyy, 80
_Dyz, 80
_Dzx, 80
_Dzy, 80
_Dzz, 80
_dispersion, 79
_index, 80
_input_molefraction, 80
_input_pressure, 80
_input_temperature, 80
_molecular_diffusion, 80
_u_input, 81
_vel, 81
_velocity, 81
_vx, 81
_vy, 81
_vz, 81
computeQpJacobian, 79
computeQpResidual, 79
DGMassFluxBC, 79
DGMassFluxBC, 79
DGMassFluxBC.h, 231
validParams< DGMassFluxBC >, 232
DGMassFluxLimitedBC, 81
_Diffusion, 83
_Dxx, 83
_Dxy, 83
_Dxz, 83
_Dyx, 83
_Dyy, 84
_Dyz, 84
_Dzx, 84
_Dzy, 84
_Dzz, 84
_dispersion, 83
_epsilon, 84
_index, 84
_input_molefraction, 84
_input_pressure, 84
_input_temperature, 84
_molecular_diffusion, 84
_sigma, 84
_u_input, 84
_vel, 85
_velocity, 85
_vx, 85
_vy, 85
_vz, 85
computeQpJacobian, 83
computeQpResidual, 83
DGMassFluxLimitedBC, 83
DGMassFluxLimitedBC, 83
DGMassFluxLimitedBC.h, 232
validParams< DGMassFluxLimitedBC >, 233
DGOSPREY_REVISION
DgospreyRevision.h, 234
dHo
GSTA_DATA, 134
DIC
FINCH_DATA, 92
dSo
GSTA_DATA, 134
Data
Matrix, 168
default_Dc
skua.h, 296
default_adsorption
scopsowl.h, 289
default_bcs
finch.h, 244
default_effective_diffusion
scopsowl.h, 289
default_execution
finch.h, 244

default_filmMassTransfer
scopsowl.h, 290
default_ic
finch.h, 244
default_kf
skua.h, 296
default_params
finch.h, 244
default_pore_diffusion
scopsowl.h, 290
default_postprocess
finch.h, 244
default_precon
finch.h, 244
default_preprocess
finch.h, 244
default_res
finch.h, 244
default_reset
finch.h, 245
default_retardation
scopsowl.h, 290
default_solve
finch.h, 245
default_surf_diffusion
scopsowl.h, 290
default_timestep
finch.h, 245
density
PURE_GAS, 185
determinate
Matrix, 162
DgospreyApp, 85
∼DgospreyApp, 86
associateSyntax, 86
DgospreyApp, 86
DgospreyApp, 86
registerApps, 86
registerObjects, 86
DgospreyApp.h, 233
validParams$<$ DgospreyApp $>$, 233
DgospreyRevision.h, 233
DGOSPREY_REVISION, 234
diagonalSolve
Matrix, 163
dim_mis_match
error.h, 239
Dirichlet
FINCH_DATA, 92
DirichletBC
SCOPSOWL_DATA, 188
SKUA_DATA, 196
dirichletBCFill
Matrix, 163
discretize
FINCH_DATA, 92
Display
Matrix, 163
Dk
scopsowl.h, 288
Dn
FINCH_DATA, 92
Dnp1
FINCH_DATA, 92
Do
FINCH_DATA, 92
Dp
scopsowl.h, 289
Dp_ij
egret.h, 236
dq_dc
SCOPSOWL_PARAM_DATA, 193
dq_dco
SCOPSOWL_PARAM_DATA, 193
dq_dp
magpie.h, 274
dt
FINCH_DATA, 92
dt_old
FINCH_DATA, 92
duplicate_variable
error.h, 240
dxj
NUM_JAC_DATA, 173
dynamic_viscosity
PURE_GAS, 185
dz
FINCH_DATA, 93

e0
GMRESRP_DATA, 130
e0_bar
GMRESRP_DATA, 130
e1
ARNOLDI_DATA, 19
EGRET_HPP_
egret.h, 236
eMax
magpie.h, 274
mSPD_DATA, 172
edit
Matrix, 163
egret.h, 234
CE3, 235
calculate_properties, 237
Cstd, 235
D_ii, 235
D_ij, 236
Dp_ij, 236
EGRET_HPP_, 236
FilmMTCoeff, 236
initialize_data, 237
Mu, 236
Nu, 236
PE3, 236
PSI, 236
Po, 236

Pstd, 236
RE3, 236
ReNum, 237
Rstd, 237
ScNum, 237
set_variables, 237

empirical_kf
skua.h, 296

empty_matrix
error.h, 239

eps
NUM_JAC_DATA, 173
PJFNK_DATA, 181

error
error.h, 240

error.h
anchor_alias_dne, 240
arg_matrix_same, 240
dim_mis_match, 239
duplicate_variable, 240
empty_matrix, 239
file_dne, 239
generic_error, 239
indexing_error, 239
initial_error, 240
invalid_atom, 240
invalid_boolean, 239
invalid_components, 239
invalid_console_input, 240
invalid_electron, 240
invalid_fraction, 239
invalid_gas_sum, 239
invalid_molefraction, 239
invalid_neutron, 240
invalid_norm, 240
invalid_proton, 240
invalid_size, 240
invalid_solid_sum, 239
invalid_species, 240
invalid_type, 240
invalid_valence, 240
key_not_found, 240
magpie_reverse_error, 239
matrix_too_small, 240
matvec_mis_match, 240
missing_information, 240
negative_mass, 239
negative_time, 239
no_diffusion, 239
non_real_edge, 240
non_square_matrix, 239
not_a_token, 240
nullptr_error, 240
nullptr_func, 240
opt_no_support, 239
ortho_check_fail, 239
out_of_bounds, 239
read_error, 240
rxn_rate_error, 240
scenario_fail, 239
simulation_fail, 239
singular_matrix, 240
string_parse_error, 240
tensor_out_of_bounds, 240
unregistered_name, 240
unstable_matrix, 239
vector_out_of_bounds, 240
zero_vector, 240

error.h, 237
error, 240
error_type, 239
mError, 239

error_type
error.h, 239

eta
mSPD_DATA, 172

eval_GPAST
magpie.h, 275

eval_ads
SCOPSOWL_DATA, 189

eval_diff
SCOPSOWL_DATA, 189
SKUA_DATA, 196

eval_eta
magpie.h, 274

eval_kf
SCOPSOWL_DATA, 189
SKUA_DATA, 196

eval_po
magpie.h, 275

eval_po_PI
magpie.h, 275

eval_po_qo
magpie.h, 276

eval_retard
SCOPSOWL_DATA, 189

eval_surfDiff
SCOPSOWL_DATA, 189

evalprecon
FINCH_DATA, 93

evalres
FINCH_DATA, 93

ExplicitFlux
FINCH_DATA, 93

F
PJFNK_DATA, 181

FINCH_Picard
finch.h, 243

FOM
lark.h, 258

fC_E
FINCH_DATA, 93

fC_I
FINCH_DATA, 93

FINCH_DATA, 86
beta, 91

CC_E, 91
CC_I, 91
CL_E, 91
CL_I, 91
CN, 91
CR_E, 92
CR_I, 92
callroutine, 91
CheckMass, 91
d, 92
DIC, 92
Dirichlet, 92
discretize, 92
Dn, 92
Dnp1, 92
Do, 92
dt, 92
dt_old, 92
dz, 93
evalprecon, 93
evalres, 93
ExplicitFlux, 93
fC_E, 93
fC_I, 93
fL_E, 93
fL_I, 93
fR_E, 93
fR_I, 94
Fn, 93
Fnp1, 93
gE, 94
gI, 94
Iterative, 94
kIC, 94
kfn, 94
kfnp1, 94
kn, 94
knp1, 94
ko, 94
L, 94
LN, 95
lambda_E, 95
lambda_I, 95
ME, 95
MI, 95
max_iter, 95
NE, 95
NI, 95
nl_method, 95
NormTrack, 95
OE, 95
OI, 96
param_data, 96
picard_dat, 96
pjfnk_dat, 96
pres, 96
RIC, 96
res, 96
resettime, 96
Rn, 96
Rnp1, 96
Ro, 96
s, 97
setbcs, 97
setic, 97
setparams, 97
setpostprocess, 97
setpreprocess, 97
settime, 97
Sn, 97
Snp1, 97
solve, 97
SteadyState, 97
T, 98
t, 98
t_old, 98
tol_abs, 98
tol_rel, 98
total_iter, 98
u_star, 98
uAvg, 98
uAvg_old, 98
uIC, 98
uT, 99
uT_old, 99
ubest, 98
un, 99
unm1, 99
unp1, 99
uo, 99
Update, 99
uz_l_E, 99
uz_l_I, 99
uz_lm1_E, 99
uz_lm1_I, 99
uz_lp1_E, 99
uz_lp1_I, 100
vIC, 100
vn, 100
vnp1, 100
vo, 100
fL_E
FINCH_DATA, 93
fL_I
FINCH_DATA, 93
fR_E
FINCH_DATA, 93
fR_I
FINCH_DATA, 94
file_dne
error.h, 239
film_transfer
SCOPSOWL_PARAM_DATA, 193
SKUA_PARAM, 199
FilmMTCoeff
egret.h, 236

finch.h
Cartesian, 243
Cylindrical, 243
FINCH_Picard, 243
LARK_PJFNK, 243
LARK_Picard, 243
Spherical, 243
finch.h, 240
check_Mass, 244
default_bcs, 244
default_execution, 244
default_ic, 244
default_params, 244
default_postprocess, 244
default_precon, 244
default_preprocess, 244
default_res, 244
default_reset, 245
default_solve, 245
default_timestep, 245
finch_coord_type, 243
finch_solve_type, 243
l_direct, 245
lark_picard_step, 245
max, 245
min, 245
minmod, 245
minmod_discretization, 245
nl_picard, 245
ospre_discretization, 246
print2file_dim_header, 246
print2file_newline, 246
print2file_result_new, 246
print2file_result_old, 246
print2file_tab, 246
print2file_time_header, 246
setup_FINCH_DATA, 246
uAverage, 247
uTotal, 247
vanAlbada_discretization, 247
finch_coord_type
finch.h, 243
finch_dat
SCOPSOWL_DATA, 189
SKUA_DATA, 197
finch_solve_type
finch.h, 243
Fk
BACKTRACK_DATA, 22
flock.h, 247
FlowProperties, 100
_binder_porosity, 103
_column_length, 103
_comp_Sutherland_const, 103
_comp_heat_capacity, 103
_comp_ref_temp, 103
_comp_ref_viscosity, 103
_dispersion, 103
_film_transfer, 103
_flow_rate, 103
_gas_conc, 103
_gas_density, 104
_gas_heat_capacity, 104
_gas_molecular_wieght, 104
_gas_viscosity, 104
_heat_retardation, 104
_index, 104
_inner_dia, 104
_mixed_gas, 104
_mixed_gas_old, 104
_molecular_diffusion, 104
_molecular_weight, 104
_pellet_density, 105
_pellet_diameter, 105
_pellet_heat_capacity, 105
_pore_diffusion, 105
_pore_size, 105
_porosity, 105
_retardation, 105
_solid_conc, 105
_solid_perturb, 105
_temperature, 105
_total_pressure, 105
_velocity, 106
computeQpProperties, 102
FlowProperties, 102
FlowProperties, 102
initQpStatefulProperties, 102
FlowProperties.h, 248
_gas_const, 249
validParams< FlowProperties >, 249
Fn
FINCH_DATA, 93
Fnp1
FINCH_DATA, 93
fom
lark.h, 261
fun_call
BACKTRACK_DATA, 22
PJFNK_DATA, 181
funeval
PJFNK_DATA, 182
Fv
PJFNK_DATA, 182
Fx
NUM_JAC_DATA, 173
Fxp
NUM_JAC_DATA, 173

GCR
lark.h, 258
GMRESLP
lark.h, 258
GMRESR
lark.h, 258
GMRESRP
lark.h, 258

GAdvection, 106
    _velocity, 107
    _vx, 107
    _vy, 107
    _vz, 107
    computeQpJacobian, 107
    computeQpResidual, 107
    GAdvection, 107
    GAdvection, 107
GAdvection.h, 249
    validParams< GAdvection >, 250
GAnisotropicDiffusion, 108
    _Diffusion, 109
    _Dxx, 109
    _Dxy, 109
    _Dxz, 109
    _Dyx, 109
    _Dyy, 110
    _Dyz, 110
    _Dzx, 110
    _Dzy, 110
    _Dzz, 110
    computeQpJacobian, 109
    computeQpResidual, 109
    GAnisotropicDiffusion, 109
    GAnisotropicDiffusion, 109
GAnisotropicDiffusion.h, 250
    validParams< GAnisotropicDiffusion >, 250
GCR_DATA, 119
    alpha, 121
    bestres, 121
    bestx, 121
    beta, 121
    breakdown, 121
    c, 121
    c_temp, 121
    iter_inner, 121
    iter_outer, 121
    maxit, 122
    Output, 122
    r, 122
    relres, 122
    relres_base, 122
    res, 122
    restart, 122
    tol_abs, 122
    tol_rel, 122
    total_iter, 122
    transpose_dat, 122
    u, 122
    u_temp, 123
    x, 123
GCR_Output
    GMRESR_DATA, 127
GColumnHeatAdvection, 110
    _gas_density, 112
    _gas_heat_capacity, 112
    _vel, 112
    _velocity, 112
    _vx, 112
    _vy, 112
    _vz, 112
    computeQpJacobian, 111
    computeQpResidual, 111
    GColumnHeatAdvection, 111
    GColumnHeatAdvection, 111
GColumnHeatAdvection.h, 250
    validParams< GColumnHeatAdvection >, 251
GColumnHeatDispersion, 112
    _Diffusion, 114
    _Dxx, 114
    _Dxy, 114
    _Dxz, 114
    _Dyx, 114
    _Dyy, 114
    _Dyz, 114
    _Dzx, 114
    _Dzy, 114
    _Dzz, 114
    _conductivity, 114
    computeQpJacobian, 114
    computeQpResidual, 114
    GColumnHeatDispersion, 113
    GColumnHeatDispersion, 113
GColumnHeatDispersion.h, 251
    validParams< GColumnHeatDispersion >, 252
GColumnMassAdvection, 115
    _vel, 116
    _velocity, 116
    _vx, 116
    _vy, 116
    _vz, 116
    computeQpJacobian, 116
    computeQpResidual, 116
    GColumnMassAdvection, 116
    GColumnMassAdvection, 116
GColumnMassAdvection.h, 252
    validParams< GColumnMassAdvection >, 253
GColumnMassDispersion, 117
    _Diffusion, 118
    _Dxx, 119
    _Dxy, 119
    _Dxz, 119
    _Dyx, 119
    _Dyy, 119
    _Dyz, 119
    _Dzx, 119
    _Dzy, 119
    _Dzz, 119
    _dispersion, 118
    _index, 119
    _molecular_diffusion, 119
    computeQpJacobian, 118
    computeQpResidual, 118
    GColumnMassDispersion, 118
    GColumnMassDispersion, 118

GColumnMassDispersion.h, 253
validParams< GColumnMassDispersion >, 254
gE
FINCH_DATA, 94
gI
FINCH_DATA, 94
GMRES_Output
GMRESR_DATA, 127
GMRESLP_DATA, 123
arnoldi_dat, 124
bestres, 124
bestx, 124
iter, 124
maxit, 124
Output, 124
r, 124
relres, 124
relres_base, 125
res, 125
restart, 125
steps, 125
tol_abs, 125
tol_rel, 125
x, 125
GMRESR_DATA, 125
arg, 126
GCR_Output, 127
GMRES_Output, 127
gcr_abs_tol, 126
gcr_dat, 127
gcr_maxit, 127
gcr_rel_tol, 127
gcr_restart, 127
gmres_dat, 127
gmres_maxit, 127
gmres_restart, 127
gmres_tol, 127
iter_inner, 127
iter_outer, 128
matvec, 128
matvec_data, 128
N, 128
term_precon, 128
terminal_precon, 128
total_iter, 128
GMRESRP_DATA, 128
bestres, 130
bestx, 130
e0, 130
e0_bar, 130
H, 130
H_bar, 130
iter_inner, 130
iter_outer, 130
iter_total, 130
maxit, 131
Output, 131
r, 131
relres, 131
relres_base, 131
res, 131
restart, 131
sum, 131
tol_abs, 131
tol_rel, 131
v, 131
Vk, 131
w, 132
x, 132
y, 132
Zk, 132
GPAST_DATA, 132
gama_inf, 133
He, 133
PIo, 133
po, 133
poi, 133
present, 133
q, 133
qo, 133
x, 133
y, 134
GSTA_DATA, 134
dHo, 134
dSo, 134
m, 134
qmax, 134
gama
mSPD_DATA, 172
gama_inf
GPAST_DATA, 133
gas_dat
SCOPSOWL_DATA, 189
SKUA_DATA, 197
gas_temperature
MIXED_GAS, 170
SCOPSOWL_DATA, 189
gas_velocity
SCOPSOWL_DATA, 189
SKUA_DATA, 197
gcr
lark.h, 261
gcr_abs_tol
GMRESR_DATA, 126
gcr_dat
GMRESR_DATA, 127
PJFNK_DATA, 182
gcr_maxit
GMRESR_DATA, 127
gcr_rel_tol
GMRESR_DATA, 127
gcr_restart
GMRESR_DATA, 127
generic_error
error.h, 239
gmres_dat

GMRESR_DATA, 127
gmres_in
KMS_DATA, 136
gmres_maxit
GMRESR_DATA, 127
gmres_out
KMS_DATA, 136
gmres_restart
GMRESR_DATA, 127
gmres_tol
GMRESR_DATA, 127
gmresLeftPreconditioned
lark.h, 262
gmresRightPreconditioned
lark.h, 263
gmreslp_dat
PJFNK_DATA, 182
gmresr
lark.h, 263
gmresr_dat
PJFNK_DATA, 182
gmresrPreconditioner
lark.h, 264
gmresrp_dat
PJFNK_DATA, 182
gpast_dat
MAGPIE_DATA, 146
grad_mSPD
magpie.h, 276
gsta_dat
MAGPIE_DATA, 146

H
GMRESRP_DATA, 130
H_bar
GMRESRP_DATA, 130
He
GPAST_DATA, 133
magpie.h, 273
Heterogeneous
SCOPSOWL_DATA, 189
Hkp1
ARNOLDI_DATA, 19
hp1
ARNOLDI_DATA, 19

I
SYSTEM_DATA, 201
Ideal
SYSTEM_DATA, 201
Ii
OPTRANS_DATA, 174
indexing_error
error.h, 239
initQpStatefulProperties
FlowProperties, 102
MagpieAdsorbateProperties, 155
initial_error
error.h, 240
initialGuess_mSPD
magpie.h, 276
initialize_data
egret.h, 237
inner_iter
KMS_DATA, 136
inner_product
Matrix, 163
inner_reltol
KMS_DATA, 136
IntegralAvg
Matrix, 163
IntegralTotal
Matrix, 164
invalid_atom
error.h, 240
invalid_boolean
error.h, 239
invalid_components
error.h, 239
invalid_console_input
error.h, 240
invalid_electron
error.h, 240
invalid_fraction
error.h, 239
invalid_gas_sum
error.h, 239
invalid_molefraction
error.h, 239
invalid_neutron
error.h, 240
invalid_norm
error.h, 240
invalid_proton
error.h, 240
invalid_size
error.h, 240
invalid_solid_sum
error.h, 239
invalid_species
error.h, 240
invalid_type
error.h, 240
invalid_valence
error.h, 240
inverse
Matrix, 164
iter
ARNOLDI_DATA, 19
BiCGSTAB_DATA, 33
CGS_DATA, 37
GMRESLP_DATA, 124
PCG_DATA, 176
PICARD_DATA, 178
iter_inner
GCR_DATA, 121
GMRESR_DATA, 127

GMRESRP_DATA, 130
iter_outer
GCR_DATA, 121
GMRESR_DATA, 128
GMRESRP_DATA, 130
iter_total
GMRESRP_DATA, 130
Iterative
FINCH_DATA, 94

J
SYSTEM_DATA, 202
jacvec
lark.h, 264

K
SYSTEM_DATA, 202
k
ARNOLDI_DATA, 19
kB
magpie.h, 273
kIC
FINCH_DATA, 94
KMS_DATA, 135
gmres_in, 136
gmres_out, 136
inner_iter, 136
inner_reltol, 136
level, 136
matvec, 136
matvec_data, 136
max_level, 136
maxit, 136
outer_abstol, 137
outer_iter, 137
outer_reltol, 137
Output_in, 137
Output_out, 137
restart, 137
term_precon, 137
terminal_precon, 137
total_iter, 137
key_not_found
error.h, 240
kfn
FINCH_DATA, 94
kfnp1
FINCH_DATA, 94
kinematic_viscosity
MIXED_GAS, 170
kmsPreconditioner
lark.h, 264
kn
FINCH_DATA, 94
knp1
FINCH_DATA, 94
ko
FINCH_DATA, 94
krylov_method
lark.h, 258
krylovMultiSpace
lark.h, 265

L
FINCH_DATA, 94
LARK_PJFNK
finch.h, 243
LARK_Picard
finch.h, 243
L_Output
PJFNK_DATA, 182
l_direct
finch.h, 245
l_iter
PJFNK_DATA, 182
LN
FINCH_DATA, 95
ladshawSolve
Matrix, 164
lambda_E
FINCH_DATA, 95
lambda_I
FINCH_DATA, 95
lambdaMin
BACKTRACK_DATA, 22
lark.h
BiCGSTAB, 258
CGS, 258
FOM, 258
GCR, 258
GMRESLP, 258
GMRESR, 258
GMRESRP, 258
PCG, 258
lark.h, 254
arnoldi, 258
backtrackLineSearch, 259
bicgstab, 259
cgs, 260
fom, 261
gcr, 261
gmresLeftPreconditioned, 262
gmresRightPreconditioned, 263
gmresr, 263
gmresrPreconditioner, 264
jacvec, 264
kmsPreconditioner, 264
krylov_method, 258
krylovMultiSpace, 265
MIN_TOL, 258
NumericalJacobian, 265
operatorTranspose, 266
pcg, 266
picard, 267
pjfnk, 267
update_arnoldi_solution, 268
lark_picard_step
finch.h, 245

level
KMS_DATA, 136
SCOPSOWL_DATA, 189
lin_tol_abs
PJFNK_DATA, 182
lin_tol_rel
PJFNK_DATA, 182
LineSearch
PJFNK_DATA, 183
linear_solver
PJFNK_DATA, 182
LinearDrivingForce, 138
_coef, 139
_driving_value, 139
_gaining, 139
_var, 139
computeQpJacobian, 139
computeQpResidual, 139
LinearDrivingForce, 139
LinearDrivingForce, 139
LinearDrivingForce.h, 269
validParams< LinearDrivingForce >, 269
lnKo
magpie.h, 273
lnact_mSPD
magpie.h, 276
lowerHessenberg2Triangular
Matrix, 164
lowerHessenbergSolve
Matrix, 164
lowerTriangularSolve
Matrix, 164

m
GSTA_DATA, 134
M_PI
macaw.h, 271
MAGPIE
magpie.h, 277
MAGPIE_Adsorption, 139
_index, 141
_magpie_dat, 141
computeValue, 140
MAGPIE_Adsorption, 140
MAGPIE_Adsorption, 140
MAGPIE_Adsorption.h, 278
validParams< MAGPIE_Adsorption >, 279
MAGPIE_AdsorptionHeat, 141
_index, 142
_magpie_dat, 142
_solid_conc, 142
computeValue, 142
MAGPIE_AdsorptionHeat, 142
MAGPIE_AdsorptionHeat, 142
MAGPIE_AdsorptionHeat.h, 279
MAGPIE_ConstLDF_Adsorption, 142
_driving_value, 144
_index, 144
_ldf_coef, 144
_magpie_dat, 144
computeValue, 144
MAGPIE_ConstLDF_Adsorption.h, 280
MAGPIE_ConstLDF_Perturbation, 144
_driving_value, 145
_index, 145
_ldf_coef, 146
_magpie_dat, 146
computeValue, 145
MAGPIE_ConstLDF_Perturbation.h, 281
MAGPIE_DATA, 146
gpast_dat, 146
gsta_dat, 146
mspd_dat, 146
sys_dat, 146
MAGPIE_MaterialLDF_Adsorption, 147
_binder_porosity, 148
_crystal_radius, 148
_driving_value, 148
_film_transfer, 148
_index, 149
_ldf_coef, 149
_magpie_dat, 149
_pellet_density, 149
_pellet_diameter, 149
_pore_diffusion, 149
_porosity, 149
_surface_diffusion, 149
computeValue, 148
MAGPIE_MaterialLDF_Adsorption.h, 282
MAGPIE_MaterialLDF_Perturbation, 149
_binder_porosity, 151
_crystal_radius, 151
_driving_value, 151
_film_transfer, 151
_index, 151
_ldf_coef, 151
_magpie_dat, 152
_pellet_density, 152
_pellet_diameter, 152
_pore_diffusion, 152
_porosity, 152
_surface_diffusion, 152
computeValue, 151
MAGPIE_MaterialLDF_Perturbation.h, 283
MAGPIE_Perturbation, 152
_index, 153
_magpie_dat, 153
computeValue, 153
MAGPIE_Perturbation, 153
MAGPIE_Perturbation, 153
MAGPIE_Perturbation.h, 284
validParams< MAGPIE_Perturbation >, 286
ME
FINCH_DATA, 95
mError
error.h, 239
MI

FINCH_DATA, 95
MIN_TOL
lark.h, 258
MIXED_GAS, 169
binary_diffusion, 170
char_length, 170
CheckMolefractions, 170
gas_temperature, 170
kinematic_viscosity, 170
molefraction, 170
N, 171
Reynolds, 171
species_dat, 171
total_density, 171
total_dyn_vis, 171
total_molecular_weight, 171
total_pressure, 171
total_specific_heat, 171
velocity, 171
mSPD_DATA, 172
eMax, 172
eta, 172
gama, 172
s, 172
v, 172
macaw.h, 269
M_PI, 271
magpie.h, 271
A, 273
DBL_EPSILON, 273
dq_dp, 274
eMax, 274
eval_GPAST, 275
eval_eta, 274
eval_po, 275
eval_po_PI, 275
eval_po_qo, 276
grad_mSPD, 276
He, 273
initialGuess_mSPD, 276
kB, 273
lnKo, 273
lnact_mSPD, 276
MAGPIE, 277
Na, 274
PI, 277
Po, 274
q_p, 277
qT, 278
qo, 278
Qst, 278
R, 274
shapeFactor, 274
V, 274
Z, 274
magpie_reverse_error
error.h, 239
magpie_dat
SCOPSOWL_DATA, 190
SKUA_DATA, 197
MagpieAdsorbateProperties, 154
_enthalpy_1, 156
_enthalpy_2, 156
_enthalpy_3, 156
_enthalpy_4, 156
_enthalpy_5, 156
_enthalpy_6, 156
_entropy_1, 156
_entropy_2, 156
_entropy_3, 156
_entropy_4, 156
_entropy_5, 157
_entropy_6, 157
_gas_conc, 157
_gas_conc_old, 157
_index, 157
_magpie_dat, 157
_magpie_dat_old, 157
_max_capacity, 157
_molar_volume, 157
_num_sites, 157
_temperature, 158
_total_pressure, 158
computeQpProperties, 155
initQpStatefulProperties, 155
MagpieAdsorbateProperties, 155
MagpieAdsorbateProperties, 155
MagpieAdsorbateProperties.h, 286
validParams< MagpieAdsorbateProperties >, 286
Matrix
~Matrix, 161
adjoint, 161
cofactor, 161
columnExtend, 161
columnExtract, 162
columnProjection, 162
columnReplace, 162
columnShrink, 162
columnVectorFill, 162
columns, 162
ConstantICFill, 162
Data, 168
determinate, 162
diagonalSolve, 163
dirichletBCFill, 163
Display, 163
edit, 163
inner_product, 163
IntegralAvg, 163
IntegralTotal, 164
inverse, 164
ladshawSolve, 164
lowerHessenberg2Triangular, 164
lowerHessenbergSolve, 164
lowerTriangularSolve, 164
Matrix, 161

naturalLaplacian3D, 165
norm, 165
num_cols, 169
num_rows, 169
operator*, 165
operator(), 165
operator+, 165
operator-, 165
operator/, 165
operator=, 166
rowExtend, 166
rowExtract, 166
rowReplace, 166
rowShrink, 166
rows, 166
set_size, 166
SolnTransform, 166
sphericalAvg, 166
sphericalBCFill, 167
sum, 167
transpose, 167
transpose_multiply, 167
tridiagonalFill, 167
tridiagonalSolve, 167
tridiagonalVectorFill, 168
upperHessenberg2Triangular, 168
upperHessenbergSolve, 168
upperTriangularSolve, 168
zeros, 168
Matrix< T >, 158
matrix_too_small
error.h, 240
matvec
GMRESR_DATA, 128
KMS_DATA, 136
matvec_mis_match
error.h, 240
matvec_data
GMRESR_DATA, 128
KMS_DATA, 136
max
finch.h, 245
max_iter
FINCH_DATA, 95
max_level
KMS_DATA, 136
max_norm
SYSTEM_DATA, 202
maxit
BiCGSTAB_DATA, 33
CGS_DATA, 37
GCR_DATA, 122
GMRESLP_DATA, 124
GMRESRP_DATA, 131
KMS_DATA, 136
PCG_DATA, 176
PICARD_DATA, 178
min
finch.h, 245
minmod
finch.h, 245
minmod_discretization
finch.h, 245
missing_information
error.h, 240
molecular_diffusion
PURE_GAS, 185
molecular_weight
PURE_GAS, 185
molefraction
MIXED_GAS, 170
molefractionCheck
skua.h, 296
mspd_dat
MAGPIE_DATA, 146
Mu
egret.h, 236

N
GMRESR_DATA, 128
MIXED_GAS, 171
SYSTEM_DATA, 202
NE
FINCH_DATA, 95
NI
FINCH_DATA, 95
NL_Output
PJFNK_DATA, 183
NUM_JAC_DATA, 173
dxj, 173
eps, 173
Fx, 173
Fxp, 173
Na
magpie.h, 274
naturalLaplacian3D
Matrix, 165
negative_mass
error.h, 239
negative_time
error.h, 239
nl_bestres
PJFNK_DATA, 183
nl_iter
PJFNK_DATA, 183
nl_maxit
PJFNK_DATA, 183
nl_method
FINCH_DATA, 95
nl_picard
finch.h, 245
nl_relres
PJFNK_DATA, 183
nl_res
PJFNK_DATA, 183
nl_res_base
PJFNK_DATA, 183

nl_tol_abs
PJFNK_DATA, 183
nl_tol_rel
PJFNK_DATA, 183
no_diffusion
error.h, 239
non_real_edge
error.h, 240
non_square_matrix
error.h, 239
NonLinear
SCOPSOWL_DATA, 190
SKUA_DATA, 197
norm
Matrix, 165
normFkp1
BACKTRACK_DATA, 22
NormTrack
FINCH_DATA, 95
not_a_token
error.h, 240
Nu
egret.h, 236
nullptr_error
error.h, 240
nullptr_func
error.h, 240
num_cols
Matrix, 169
num_rows
Matrix, 169
NumericalJacobian
lark.h, 265

OE
FINCH_DATA, 95
OI
FINCH_DATA, 96
OPTRANS_DATA, 174
Ai, 174
Ii, 174
omega
BiCGSTAB_DATA, 33
omega_old
BiCGSTAB_DATA, 33
operator*
Matrix, 165
operator()
Matrix, 165
operator+
Matrix, 165
operator-
Matrix, 165
operator/
Matrix, 165
operator=
Matrix, 166
operatorTranspose
lark.h, 266
opt_no_support
error.h, 239
ortho_check_fail
error.h, 239
ospre_discretization
finch.h, 246
out_of_bounds
error.h, 239
outer_abstol
KMS_DATA, 137
outer_iter
KMS_DATA, 137
outer_reltol
KMS_DATA, 137
Output
ARNOLDI_DATA, 19
BiCGSTAB_DATA, 33
CGS_DATA, 37
GCR_DATA, 122
GMRESLP_DATA, 124
GMRESRP_DATA, 131
PCG_DATA, 176
PICARD_DATA, 178
SYSTEM_DATA, 202
Output_in
KMS_DATA, 137
Output_out
KMS_DATA, 137
OutputFile
SCOPSOWL_DATA, 190
SKUA_DATA, 197

p
BiCGSTAB_DATA, 33
CGS_DATA, 37
PCG_DATA, 176
PCG
lark.h, 258
PCG_DATA, 174
alpha, 175
Ap, 175
bestres, 175
bestx, 176
beta, 176
iter, 176
maxit, 176
Output, 176
p, 176
r, 176
r_old, 176
relres, 176
relres_base, 176
res, 176
tol_abs, 177
tol_rel, 177
x, 177
z, 177
z_old, 177
PE3

egret.h, 236
PI
magpie.h, 277
SYSTEM_DATA, 202
PICARD_DATA, 177
bestres, 178
bestx, 178
iter, 178
maxit, 178
Output, 178
r, 178
relres, 178
relres_base, 179
res, 179
tol_abs, 179
tol_rel, 179
x0, 179
PIo
GPAST_DATA, 133
PJFNK_DATA, 179
backtrack_dat, 181
bestx, 181
bicgstab_dat, 181
Bounce, 181
cgs_dat, 181
eps, 181
F, 181
fun_call, 181
funeval, 182
Fv, 182
gcr_dat, 182
gmreslp_dat, 182
gmresr_dat, 182
gmresrp_dat, 182
L_Output, 182
l_iter, 182
lin_tol_abs, 182
lin_tol_rel, 182
LineSearch, 183
linear_solver, 182
NL_Output, 183
nl_bestres, 183
nl_iter, 183
nl_maxit, 183
nl_relres, 183
nl_res, 183
nl_res_base, 183
nl_tol_abs, 183
nl_tol_rel, 183
pcg_dat, 183
precon, 184
precon_data, 184
res_data, 184
v, 184
x, 184
PSI
egret.h, 236
PT
SYSTEM_DATA, 202
PURE_GAS, 184
density, 185
dynamic_viscosity, 185
molecular_diffusion, 185
molecular_weight, 185
Schmidt, 185
specific_heat, 185
Sutherland_Const, 185
Sutherland_Temp, 185
Sutherland_Viscosity, 186
Par
SYSTEM_DATA, 202
param_dat
SCOPSOWL_DATA, 190
SKUA_DATA, 197
param_data
FINCH_DATA, 96
pcg
lark.h, 266
pcg_dat
PJFNK_DATA, 183
pellet_density
SCOPSOWL_DATA, 190
pellet_radius
SCOPSOWL_DATA, 190
SKUA_DATA, 197
pi
SYSTEM_DATA, 202
picard
lark.h, 267
picard_dat
FINCH_DATA, 96
pjfnk
lark.h, 267
pjfnk_dat
FINCH_DATA, 96
Po
egret.h, 236
magpie.h, 274
po
GPAST_DATA, 133
poi
GPAST_DATA, 133
pore_diffusion
SCOPSOWL_PARAM_DATA, 193
precon
PJFNK_DATA, 184
precon_data
PJFNK_DATA, 184
pres
FINCH_DATA, 96
present
GPAST_DATA, 133
Print2Console
SCOPSOWL_DATA, 190
SKUA_DATA, 197
Print2File

SCOPSOWL_DATA, 190
SKUA_DATA, 197
print2file_SCOPSOWL_header
scopsowl.h, 291
print2file_SCOPSOWL_result_new
scopsowl.h, 291
print2file_SCOPSOWL_result_old
scopsowl.h, 291
print2file_SCOPSOWL_time_header
scopsowl.h, 291
print2file_SKUA_header
skua.h, 296
print2file_SKUA_results_new
skua.h, 297
print2file_SKUA_results_old
skua.h, 297
print2file_SKUA_time_header
skua.h, 297
print2file_dim_header
finch.h, 246
print2file_newline
finch.h, 246
print2file_result_new
finch.h, 246
print2file_result_old
finch.h, 246
print2file_species_header
scopsowl.h, 291
skua.h, 297
print2file_tab
finch.h, 246
print2file_time_header
finch.h, 246
Pstd
egret.h, 236

q
GPAST_DATA, 133
q_p
magpie.h, 277
qAvg
SCOPSOWL_PARAM_DATA, 193
qAvg_old
SCOPSOWL_PARAM_DATA, 193
qIntegralAvg
SCOPSOWL_PARAM_DATA, 193
qIntegralAvg_old
SCOPSOWL_PARAM_DATA, 194
qT
magpie.h, 278
SYSTEM_DATA, 202
qTn
SKUA_DATA, 197
qTnp1
SKUA_DATA, 198
qmax
GSTA_DATA, 134
qo
GPAST_DATA, 133
magpie.h, 278
SCOPSOWL_PARAM_DATA, 194
Qst
magpie.h, 278
SCOPSOWL_PARAM_DATA, 194
Qst_old
SCOPSOWL_PARAM_DATA, 194
QstAvg
SCOPSOWL_PARAM_DATA, 194
QstAvg_old
SCOPSOWL_PARAM_DATA, 194
Qstn
SKUA_PARAM, 199
Qstnp1
SKUA_PARAM, 199
Qsto
SCOPSOWL_PARAM_DATA, 194

R
magpie.h, 274
r
BiCGSTAB_DATA, 33
CGS_DATA, 37
GCR_DATA, 122
GMRESLP_DATA, 124
GMRESRP_DATA, 131
PCG_DATA, 176
PICARD_DATA, 178
r0
BiCGSTAB_DATA, 33
CGS_DATA, 37
r_old
PCG_DATA, 176
RE3
egret.h, 236
RIC
FINCH_DATA, 96
ReNum
egret.h, 237
read_error
error.h, 240
Recover
SYSTEM_DATA, 202
ref_diffusion
SCOPSOWL_PARAM_DATA, 194
SKUA_PARAM, 199
ref_pressure
SCOPSOWL_PARAM_DATA, 194
SKUA_PARAM, 200
ref_temperature
SCOPSOWL_PARAM_DATA, 194
SKUA_PARAM, 200
registerApps
DgospreyApp, 86
registerObjects
DgospreyApp, 86
relres
BiCGSTAB_DATA, 34
CGS_DATA, 37

GCR_DATA, 122
GMRESLP_DATA, 124
GMRESRP_DATA, 131
PCG_DATA, 176
PICARD_DATA, 178
relres_base
BiCGSTAB_DATA, 34
CGS_DATA, 37
GCR_DATA, 122
GMRESLP_DATA, 125
GMRESRP_DATA, 131
PCG_DATA, 176
PICARD_DATA, 179
res
BiCGSTAB_DATA, 34
CGS_DATA, 37
FINCH_DATA, 96
GCR_DATA, 122
GMRESLP_DATA, 125
GMRESRP_DATA, 131
PCG_DATA, 176
PICARD_DATA, 179
res_data
PJFNK_DATA, 184
resettime
FINCH_DATA, 96
restart
GCR_DATA, 122
GMRESLP_DATA, 125
GMRESRP_DATA, 131
KMS_DATA, 137
Reynolds
MIXED_GAS, 171
rho
BACKTRACK_DATA, 22
BiCGSTAB_DATA, 34
CGS_DATA, 37
rho_old
BiCGSTAB_DATA, 34
Rn
FINCH_DATA, 96
Rnp1
FINCH_DATA, 96
Ro
FINCH_DATA, 96
rowExtend
Matrix, 166
rowExtract
Matrix, 166
rowReplace
Matrix, 166
rowShrink
Matrix, 166
rows
Matrix, 166
Rstd
egret.h, 237
rxn_rate_error
error.h, 240
s
BiCGSTAB_DATA, 34
FINCH_DATA, 97
mSPD_DATA, 172
SCOPSOWL
scopsowl.h, 291
SCOPSOWL_DATA, 186
binder_fraction, 188
binder_poresize, 188
binder_porosity, 188
char_macro, 188
char_micro, 188
coord_macro, 188
coord_micro, 188
crystal_radius, 188
DirichletBC, 188
eval_ads, 189
eval_diff, 189
eval_kf, 189
eval_retard, 189
eval_surfDiff, 189
finch_dat, 189
gas_dat, 189
gas_temperature, 189
gas_velocity, 189
Heterogeneous, 189
level, 189
magpie_dat, 190
NonLinear, 190
OutputFile, 190
param_dat, 190
pellet_density, 190
pellet_radius, 190
Print2Console, 190
Print2File, 190
sim_time, 190
skua_dat, 190
SurfDiff, 190
t, 191
t_counter, 191
t_old, 191
t_print, 191
tempy, 191
total_pressure, 191
total_steps, 191
user_data, 191
y, 191
SCOPSOWL_Executioner
scopsowl.h, 291
SCOPSOWL_HPP_
scopsowl.h, 289
SCOPSOWL_PARAM_DATA, 192
Adsorbable, 193
affinity, 193
qAvg, 193
qo, 194
Qst, 194

QstAvg, 194
Qsto, 194
speciesName, 194
SCOPSOWL_postprocesses
scopsowl.h, 291
SCOPSOWL_preprocesses
scopsowl.h, 291
SCOPSOWL_reset
scopsowl.h, 292
SKUA
skua.h, 298
SKUA_DATA, 195
char_measure, 196
coord, 196
DirichletBC, 196
eval_diff, 196
eval_kf, 196
finch_dat, 197
gas_dat, 197
gas_velocity, 197
magpie_dat, 197
NonLinear, 197
OutputFile, 197
param_dat, 197
pellet_radius, 197
Print2Console, 197
Print2File, 197
qTn, 197
qTnp1, 198
sim_time, 198
t, 198
t_counter, 198
t_old, 198
t_print, 198
total_steps, 198
user_data, 198
y, 198
SKUA_Executioner
skua.h, 298
SKUA_HPP_
skua.h, 295
SKUA_PARAM, 199
activation_energy, 199
Adsorbable, 199
affinity, 199
film_transfer, 199
Qstn, 199
Qstnp1, 199
ref_diffusion, 199
ref_pressure, 200
ref_temperature, 200
speciesName, 200
xIC, 200
xn, 200
xnp1, 200
y_eff, 200
SKUA_postprocesses
skua.h, 298
SKUA_preprocesses
skua.h, 298
SKUA_reset
skua.h, 298
SYSTEM_DATA, 200
As, 201
avg_norm, 201
Carrier, 201
I, 201
Ideal, 201
J, 202
K, 202
max_norm, 202
N, 202
Output, 202
PI, 202
PT, 202
Par, 202
pi, 202
qT, 202
Recover, 202
Sys, 203
T, 203
total_eval, 203
ScNum
egret.h, 237
scenario_fail
error.h, 239
Schmidt
PURE_GAS, 185
scopsowl.h, 286
avgDp, 288
const_filmMassTransfer, 289
const_pore_diffusion, 289
default_adsorption, 289
default_effective_diffusion, 289
default_filmMassTransfer, 290
default_pore_diffusion, 290
default_retardation, 290
default_surf_diffusion, 290
Dk, 288
Dp, 289
print2file_SCOPSOWL_header, 291
print2file_SCOPSOWL_result_new, 291
print2file_SCOPSOWL_result_old, 291
print2file_SCOPSOWL_time_header, 291
print2file_species_header, 291
SCOPSOWL, 291
SCOPSOWL_Executioner, 291
SCOPSOWL_HPP_, 289
SCOPSOWL_postprocesses, 291
SCOPSOWL_preprocesses, 291
SCOPSOWL_reset, 292
set_SCOPSOWL_ICs, 292
set_SCOPSOWL_params, 292
set_SCOPSOWL_timestep, 292
setup_SCOPSOWL_DATA, 292
set_SCOPSOWL_ICs

scopsowl.h, 292
set_SCOPSOWL_params
scopsowl.h, 292
set_SCOPSOWL_timestep
scopsowl.h, 292
set_SKUA_ICs
skua.h, 297
set_SKUA_params
skua.h, 297
set_SKUA_timestep
skua.h, 297
set_size
Matrix, 166
set_variables
egret.h, 237
setbcs
FINCH_DATA, 97
setic
FINCH_DATA, 97
setparams
FINCH_DATA, 97
setpostprocess
FINCH_DATA, 97
setpreprocess
FINCH_DATA, 97
settime
FINCH_DATA, 97
setup_FINCH_DATA
finch.h, 246
setup_SCOPSOWL_DATA
scopsowl.h, 292
setup_SKUA_DATA
skua.h, 297
shapeFactor
magpie.h, 274
sigma
CGS_DATA, 38
sim_time
SCOPSOWL_DATA, 190
SKUA_DATA, 198
simple_darken_Dc
skua.h, 297
simulation_fail
error.h, 239
singular_matrix
error.h, 240
skua.h, 293
const_Dc, 295
const_kf, 296
D_c, 295
D_inf, 295
D_o, 295
default_Dc, 296
default_kf, 296
empirical_kf, 296
molefractionCheck, 296
print2file_SKUA_header, 296
print2file_SKUA_results_new, 297
print2file_SKUA_results_old, 297
print2file_SKUA_time_header, 297
print2file_species_header, 297
SKUA, 298
SKUA_Executioner, 298
SKUA_HPP_, 295
SKUA_postprocesses, 298
SKUA_preprocesses, 298
SKUA_reset, 298
set_SKUA_ICs, 297
set_SKUA_params, 297
set_SKUA_timestep, 297
setup_SKUA_DATA, 297
simple_darken_Dc, 297
theoretical_darken_Dc, 298
skua_dat
SCOPSOWL_DATA, 190
Sn
FINCH_DATA, 97
Snp1
FINCH_DATA, 97
SolnTransform
Matrix, 166
solve
FINCH_DATA, 97
species_dat
MIXED_GAS, 171
speciesName
SCOPSOWL_PARAM_DATA, 194
SKUA_PARAM, 200
specific_heat
PURE_GAS, 185
Spherical
finch.h, 243
sphericalAvg
Matrix, 166
sphericalBCFill
Matrix, 167
SteadyState
FINCH_DATA, 97
steps
GMRESLP_DATA, 125
string_parse_error
error.h, 240
sum
ARNOLDI_DATA, 19
GMRESRP_DATA, 131
Matrix, 167
SurfDiff
SCOPSOWL_DATA, 190
Sutherland_Const
PURE_GAS, 185
Sutherland_Temp
PURE_GAS, 185
Sutherland_Viscosity
PURE_GAS, 186
Sys
SYSTEM_DATA, 203

sys_dat
    MAGPIE_DATA, 146

T
    FINCH_DATA, 98
    SYSTEM_DATA, 203
t
    BiCGSTAB_DATA, 34
    FINCH_DATA, 98
    SCOPSOWL_DATA, 191
    SKUA_DATA, 198
t_counter
    SCOPSOWL_DATA, 191
    SKUA_DATA, 198
t_old
    FINCH_DATA, 98
    SCOPSOWL_DATA, 191
    SKUA_DATA, 198
t_print
    SCOPSOWL_DATA, 191
    SKUA_DATA, 198
tempy
    SCOPSOWL_DATA, 191
tensor_out_of_bounds
    error.h, 240
term_precon
    GMRESR_DATA, 128
    KMS_DATA, 137
terminal_precon
    GMRESR_DATA, 128
    KMS_DATA, 137
theoretical_darken_Dc
    skua.h, 298
tol_abs
    BiCGSTAB_DATA, 34
    CGS_DATA, 38
    FINCH_DATA, 98
    GCR_DATA, 122
    GMRESLP_DATA, 125
    GMRESRP_DATA, 131
    PCG_DATA, 177
    PICARD_DATA, 179
tol_rel
    BiCGSTAB_DATA, 34
    CGS_DATA, 38
    FINCH_DATA, 98
    GCR_DATA, 122
    GMRESLP_DATA, 125
    GMRESRP_DATA, 131
    PCG_DATA, 177
    PICARD_DATA, 179
total_density
    MIXED_GAS, 171
total_dyn_vis
    MIXED_GAS, 171
total_eval
    SYSTEM_DATA, 203
total_iter
    FINCH_DATA, 98
    GCR_DATA, 122
    GMRESR_DATA, 128
    KMS_DATA, 137
total_molecular_weight
    MIXED_GAS, 171
total_pressure
    MIXED_GAS, 171
    SCOPSOWL_DATA, 191
total_specific_heat
    MIXED_GAS, 171
total_steps
    SCOPSOWL_DATA, 191
    SKUA_DATA, 198
TotalColumnPressure, 203
    _gas_conc, 204
    _index, 204
    _temperature, 204
    computeValue, 204
    TotalColumnPressure, 204
    TotalColumnPressure, 204
TotalColumnPressure.h, 299
    validParams< TotalColumnPressure >, 299
TotalPressureIC, 204
    _PT_IC, 205
    TotalPressureIC, 205
    TotalPressureIC, 205
    value, 205
TotalPressureIC.h, 299
    validParams< TotalPressureIC >, 300
transpose
    Matrix, 167
transpose_dat
    GCR_DATA, 122
transpose_multiply
    Matrix, 167
tridiagonalFill
    Matrix, 167
tridiagonalSolve
    Matrix, 167
tridiagonalVectorFill
    Matrix, 168

u
    CGS_DATA, 38
    GCR_DATA, 122
u_star
    FINCH_DATA, 98
u_temp
    GCR_DATA, 123
uAverage
    finch.h, 247
uAvg
    FINCH_DATA, 98
uAvg_old
    FINCH_DATA, 98
uIC
    FINCH_DATA, 98
uT
    FINCH_DATA, 99

uT_old
 FINCH_DATA, 99
uTotal
 finch.h, 247
ubest
 FINCH_DATA, 98
un
 FINCH_DATA, 99
unm1
 FINCH_DATA, 99
unp1
 FINCH_DATA, 99
unregistered_name
 error.h, 240
unstable_matrix
 error.h, 239
uo
 FINCH_DATA, 99
Update
 FINCH_DATA, 99
update_arnoldi_solution
 lark.h, 268
upperHessenberg2Triangular
 Matrix, 168
upperHessenbergSolve
 Matrix, 168
upperTriangularSolve
 Matrix, 168
user_data
 SCOPSOWL_DATA, 191
 SKUA_DATA, 198
uz_l_E
 FINCH_DATA, 99
uz_l_I
 FINCH_DATA, 99
uz_lm1_E
 FINCH_DATA, 99
uz_lm1_I
 FINCH_DATA, 99
uz_lp1_E
 FINCH_DATA, 99
uz_lp1_I
 FINCH_DATA, 100

V
 magpie.h, 274
v
 ARNOLDI_DATA, 20
 BiCGSTAB_DATA, 34
 CGS_DATA, 38
 GMRESRP_DATA, 131
 mSPD_DATA, 172
 PJFNK_DATA, 184
vIC
 FINCH_DATA, 100
validParams< AdsorbentProperties >
 AdsorbentProperties.h, 210
validParams< AdsorptionHeatAccumulation >
 AdsorptionHeatAccumulation.h, 211
validParams< AdsorptionMassTransfer >
 AdsorptionMassTransfer.h, 212
validParams< Aux_LDF >
 Aux_LDF.h, 213
validParams< BedHeatAccumulation >
 BedHeatAccumulation.h, 214
validParams< BedMassAccumulation >
 BedMassAccumulation.h, 215
validParams< BedProperties >
 BedProperties.h, 215
validParams< BedWallHeatTransfer >
 BedWallHeatTransfer.h, 216
validParams< ColumnTemperatureIC >
 ColumnTemperatureIC.h, 217
validParams< ConcentrationIC >
 ConcentrationIC.h, 218
validParams< CoupledLDF >
 CoupledLDF.h, 219
validParams< DGAdvection >
 DGAdvection.h, 220
validParams< DGAnisotropicDiffusion >
 DGAnisotropicDiffusion.h, 221
validParams< DGColumnHeatAdvection >
 DGColumnHeatAdvection.h, 222
validParams< DGColumnHeatDispersion >
 DGColumnHeatDispersion.h, 223
validParams< DGColumnMassAdvection >
 DGColumnMassAdvection.h, 224
validParams< DGColumnMassDispersion >
 DGColumnMassDispersion.h, 225
validParams< DGColumnWallHeatFluxBC >
 DGColumnWallHeatFluxBC.h, 226
validParams< DGColumnWallHeatFluxLimitedBC >
 DGColumnWallHeatFluxLimitedBC.h, 227
validParams< DGFluxBC >
 DGFluxBC.h, 228
validParams< DGFluxLimitedBC >
 DGFluxLimitedBC.h, 229
validParams< DGHeatFluxBC >
 DGHeatFluxBC.h, 230
validParams< DGHeatFluxLimitedBC >
 DGHeatFluxLimitedBC.h, 231
validParams< DGMassFluxBC >
 DGMassFluxBC.h, 232
validParams< DGMassFluxLimitedBC >
 DGMassFluxLimitedBC.h, 233
validParams< DgospreyApp >
 DgospreyApp.h, 233
validParams< FlowProperties >
 FlowProperties.h, 249
validParams< GAdvection >
 GAdvection.h, 250
validParams< GAnisotropicDiffusion >
 GAnisotropicDiffusion.h, 250
validParams< GColumnHeatAdvection >
 GColumnHeatAdvection.h, 251
validParams< GColumnHeatDispersion >
 GColumnHeatDispersion.h, 252

validParams< GColumnMassAdvection >
GColumnMassAdvection.h, 253
validParams< GColumnMassDispersion >
GColumnMassDispersion.h, 254
validParams< LinearDrivingForce >
LinearDrivingForce.h, 269
validParams< MAGPIE_Adsorption >
MAGPIE_Adsorption.h, 279
validParams< MAGPIE_AdsorptionHeat >
MAGPIE_AdsorptionHeat.h, 280
validParams< MAGPIE_Perturbation >
MAGPIE_Perturbation.h, 286
validParams< MagpieAdsorbateProperties >
MagpieAdsorbateProperties.h, 286
validParams< TotalColumnPressure >
TotalColumnPressure.h, 299
validParams< TotalPressureIC >
TotalPressureIC.h, 300
validParams< WallAmbientHeatTransfer >
WallAmbientHeatTransfer.h, 301
validParams< WallHeatAccumulation >
WallHeatAccumulation.h, 302
value
ColumnTemperatureIC, 39
ConcentrationIC, 41
TotalPressureIC, 205
vanAlbada_discretization
finch.h, 247
vector_out_of_bounds
error.h, 240
velocity
MIXED_GAS, 171
Vk
ARNOLDI_DATA, 20
GMRESRP_DATA, 131
vn
FINCH_DATA, 100
vnp1
FINCH_DATA, 100
vo
FINCH_DATA, 100

w
ARNOLDI_DATA, 20
CGS_DATA, 38
GMRESRP_DATA, 132
WallAmbientHeatTransfer, 206
_ambient_temp, 207
_inner_dia, 207
_outer_dia, 207
_wall_exterior_transfer_coeff, 207
computeQpJacobian, 207
computeQpResidual, 207
WallAmbientHeatTransfer, 207
WallAmbientHeatTransfer, 207
WallAmbientHeatTransfer.h, 300
validParams< WallAmbientHeatTransfer >, 301
WallHeatAccumulation, 207
_wall_density, 209
_wall_heat_capacity, 209
computeQpJacobian, 208
computeQpResidual, 208
WallHeatAccumulation, 208
WallHeatAccumulation, 208
WallHeatAccumulation.h, 301
validParams< WallHeatAccumulation >, 302

x
BiCGSTAB_DATA, 34
CGS_DATA, 38
GCR_DATA, 123
GMRESLP_DATA, 125
GMRESRP_DATA, 132
GPAST_DATA, 133
PCG_DATA, 177
PJFNK_DATA, 184
x0
PICARD_DATA, 179
xIC
SCOPSOWL_PARAM_DATA, 195
SKUA_PARAM, 200
xk
BACKTRACK_DATA, 23
xn
SKUA_PARAM, 200
xnp1
SKUA_PARAM, 200

y
BiCGSTAB_DATA, 35
GMRESRP_DATA, 132
GPAST_DATA, 134
SCOPSOWL_DATA, 191
SKUA_DATA, 198
y_eff
SKUA_PARAM, 200
yk
ARNOLDI_DATA, 20

Z
magpie.h, 274
z
BiCGSTAB_DATA, 35
CGS_DATA, 38
PCG_DATA, 177
z_old
PCG_DATA, 177
zero_vector
error.h, 240
zeros
Matrix, 168
Zk
GMRESRP_DATA, 132