

**HEWES : Heisenberg-Euler Weak-Field Expansion Simulator**

v0.2.1

Generated by Doxygen 1.9.5



---

<b>1 HEWES: Heisenberg-Euler Weak-Field Expansion Simulator</b>	<b>1</b>
1.1 Contents . . . . .	1
1.2 Preparing the Makefile . . . . .	2
1.3 Short User Manual . . . . .	2
1.3.1 Note on Simulation Settings . . . . .	3
1.3.2 Note on Resource Occupation . . . . .	3
1.3.3 Note on Output Analysis . . . . .	4
1.4 Authors . . . . .	4
<b>2 Hierarchical Index</b>	<b>5</b>
2.1 Class Hierarchy . . . . .	5
<b>3 Data Structure Index</b>	<b>7</b>
3.1 Data Structures . . . . .	7
<b>4 File Index</b>	<b>9</b>
4.1 File List . . . . .	9
<b>5 Data Structure Documentation</b>	<b>11</b>
5.1 Gauss1D Class Reference . . . . .	11
5.1.1 Detailed Description . . . . .	12
5.1.2 Constructor & Destructor Documentation . . . . .	12
5.1.2.1 Gauss1D() . . . . .	12
5.1.3 Member Function Documentation . . . . .	13
5.1.3.1 addToSpace() . . . . .	13
5.1.4 Field Documentation . . . . .	14
5.1.4.1 kx . . . . .	14
5.1.4.2 ky . . . . .	14
5.1.4.3 kz . . . . .	14
5.1.4.4 phig . . . . .	14
5.1.4.5 phix . . . . .	15
5.1.4.6 phiy . . . . .	15
5.1.4.7 phiz . . . . .	15
5.1.4.8 px . . . . .	15
5.1.4.9 py . . . . .	16
5.1.4.10 pz . . . . .	16
5.1.4.11 x0x . . . . .	16
5.1.4.12 x0y . . . . .	16
5.1.4.13 x0z . . . . .	17
5.2 Gauss2D Class Reference . . . . .	17
5.2.1 Detailed Description . . . . .	18
5.2.2 Constructor & Destructor Documentation . . . . .	18
5.2.2.1 Gauss2D() . . . . .	18
5.2.3 Member Function Documentation . . . . .	19

---

5.2.3.1 addToSpace() . . . . .	19
5.2.4 Field Documentation . . . . .	20
5.2.4.1 A1 . . . . .	20
5.2.4.2 A2 . . . . .	20
5.2.4.3 Amp . . . . .	20
5.2.4.4 axis . . . . .	21
5.2.4.5 dis . . . . .	21
5.2.4.6 lambda . . . . .	21
5.2.4.7 Ph0 . . . . .	21
5.2.4.8 PhA . . . . .	22
5.2.4.9 phip . . . . .	22
5.2.4.10 w0 . . . . .	22
5.2.4.11 zr . . . . .	22
5.3 Gauss3D Class Reference . . . . .	23
5.3.1 Detailed Description . . . . .	24
5.3.2 Constructor & Destructor Documentation . . . . .	24
5.3.2.1 Gauss3D() . . . . .	24
5.3.3 Member Function Documentation . . . . .	25
5.3.3.1 addToSpace() . . . . .	25
5.3.4 Field Documentation . . . . .	25
5.3.4.1 A1 . . . . .	26
5.3.4.2 A2 . . . . .	26
5.3.4.3 Amp . . . . .	26
5.3.4.4 axis . . . . .	26
5.3.4.5 dis . . . . .	27
5.3.4.6 lambda . . . . .	27
5.3.4.7 Ph0 . . . . .	27
5.3.4.8 PhA . . . . .	27
5.3.4.9 phip . . . . .	28
5.3.4.10 w0 . . . . .	28
5.3.4.11 zr . . . . .	28
5.4 gaussian1D Struct Reference . . . . .	28
5.4.1 Detailed Description . . . . .	29
5.4.2 Field Documentation . . . . .	29
5.4.2.1 k . . . . .	29
5.4.2.2 p . . . . .	29
5.4.2.3 phi . . . . .	29
5.4.2.4 phig . . . . .	30
5.4.2.5 x0 . . . . .	30
5.5 gaussian2D Struct Reference . . . . .	30
5.5.1 Detailed Description . . . . .	30
5.5.2 Field Documentation . . . . .	31

---

5.5.2.1 amp . . . . .	31
5.5.2.2 axis . . . . .	31
5.5.2.3 ph0 . . . . .	31
5.5.2.4 phA . . . . .	31
5.5.2.5 phiP . . . . .	31
5.5.2.6 w0 . . . . .	32
5.5.2.7 x0 . . . . .	32
5.5.2.8 zr . . . . .	32
5.6 gaussian3D Struct Reference . . . . .	32
5.6.1 Detailed Description . . . . .	33
5.6.2 Field Documentation . . . . .	33
5.6.2.1 amp . . . . .	33
5.6.2.2 axis . . . . .	33
5.6.2.3 ph0 . . . . .	33
5.6.2.4 phA . . . . .	33
5.6.2.5 phiP . . . . .	34
5.6.2.6 w0 . . . . .	34
5.6.2.7 x0 . . . . .	34
5.6.2.8 zr . . . . .	34
5.7 ICSetter Class Reference . . . . .	34
5.7.1 Detailed Description . . . . .	35
5.7.2 Member Function Documentation . . . . .	35
5.7.2.1 add() . . . . .	36
5.7.2.2 addGauss1D() . . . . .	36
5.7.2.3 addGauss2D() . . . . .	37
5.7.2.4 addGauss3D() . . . . .	38
5.7.2.5 addPlaneWave1D() . . . . .	38
5.7.2.6 addPlaneWave2D() . . . . .	39
5.7.2.7 addPlaneWave3D() . . . . .	40
5.7.2.8 eval() . . . . .	40
5.7.3 Field Documentation . . . . .	41
5.7.3.1 gauss1Ds . . . . .	41
5.7.3.2 gauss2Ds . . . . .	41
5.7.3.3 gauss3Ds . . . . .	42
5.7.3.4 planeWaves1D . . . . .	42
5.7.3.5 planeWaves2D . . . . .	42
5.7.3.6 planeWaves3D . . . . .	42
5.8 Lattice Class Reference . . . . .	43
5.8.1 Detailed Description . . . . .	44
5.8.2 Constructor & Destructor Documentation . . . . .	44
5.8.2.1 Lattice() . . . . .	45
5.8.3 Member Function Documentation . . . . .	45

5.8.3.1 get_dataPointDimension() . . . . .	45
5.8.3.2 get_dx() . . . . .	46
5.8.3.3 get_dy() . . . . .	46
5.8.3.4 get_dz() . . . . .	46
5.8.3.5 get_ghostLayerWidth() . . . . .	47
5.8.3.6 get_stencilOrder() . . . . .	47
5.8.3.7 get_tot_lx() . . . . .	48
5.8.3.8 get_tot_ly() . . . . .	48
5.8.3.9 get_tot_lz() . . . . .	49
5.8.3.10 get_tot_noDP() . . . . .	49
5.8.3.11 get_tot_noP() . . . . .	49
5.8.3.12 get_tot_nx() . . . . .	50
5.8.3.13 get_tot_ny() . . . . .	50
5.8.3.14 get_tot_nz() . . . . .	50
5.8.3.15 initializeCommunicator() . . . . .	51
5.8.3.16 setDiscreteDimensions() . . . . .	51
5.8.3.17 setPhysicalDimensions() . . . . .	52
5.8.4 Field Documentation . . . . .	53
5.8.4.1 comm . . . . .	53
5.8.4.2 dataPointDimension . . . . .	53
5.8.4.3 dx . . . . .	53
5.8.4.4 dy . . . . .	53
5.8.4.5 dz . . . . .	54
5.8.4.6 ghostLayerWidth . . . . .	54
5.8.4.7 my_prc . . . . .	54
5.8.4.8 n_prc . . . . .	54
5.8.4.9 statusFlags . . . . .	55
5.8.4.10 stencilOrder . . . . .	55
5.8.4.11 sunctx . . . . .	55
5.8.4.12 tot_lx . . . . .	55
5.8.4.13 tot_ly . . . . .	56
5.8.4.14 tot_lz . . . . .	56
5.8.4.15 tot_noDP . . . . .	56
5.8.4.16 tot_noP . . . . .	56
5.8.4.17 tot_nx . . . . .	57
5.8.4.18 tot_ny . . . . .	57
5.8.4.19 tot_nz . . . . .	57
5.9 LatticePatch Class Reference . . . . .	57
5.9.1 Detailed Description . . . . .	60
5.9.2 Constructor & Destructor Documentation . . . . .	60
5.9.2.1 LatticePatch() . . . . .	61
5.9.2.2 ~LatticePatch() . . . . .	61

---

5.9.3 Member Function Documentation . . . . .	61
5.9.3.1 checkFlag() . . . . .	62
5.9.3.2 derive() . . . . .	63
5.9.3.3 derotate() . . . . .	68
5.9.3.4 discreteSize() . . . . .	70
5.9.3.5 exchangeGhostCells() . . . . .	71
5.9.3.6 generateTranslocationLookup() . . . . .	73
5.9.3.7 getDelta() . . . . .	75
5.9.3.8 initializeBuffers() . . . . .	76
5.9.3.9 origin() . . . . .	77
5.9.3.10 rotateIntoEigen() . . . . .	78
5.9.3.11 rotateToX() . . . . .	79
5.9.3.12 rotateToY() . . . . .	80
5.9.3.13 rotateToZ() . . . . .	81
5.9.4 Friends And Related Function Documentation . . . . .	82
5.9.4.1 generatePatchwork . . . . .	82
5.9.5 Field Documentation . . . . .	84
5.9.5.1 buffData . . . . .	84
5.9.5.2 buffX . . . . .	84
5.9.5.3 buffY . . . . .	84
5.9.5.4 buffZ . . . . .	84
5.9.5.5 du . . . . .	85
5.9.5.6 duData . . . . .	85
5.9.5.7 duLocal . . . . .	85
5.9.5.8 dx . . . . .	85
5.9.5.9 dy . . . . .	86
5.9.5.10 dz . . . . .	86
5.9.5.11 envelopeLattice . . . . .	86
5.9.5.12 gCLData . . . . .	86
5.9.5.13 gCRData . . . . .	87
5.9.5.14 ghostCellLeft . . . . .	87
5.9.5.15 ghostCellLeftToSend . . . . .	87
5.9.5.16 ghostCellRight . . . . .	87
5.9.5.17 ghostCellRightToSend . . . . .	88
5.9.5.18 ghostCells . . . . .	88
5.9.5.19 ghostCellsToSend . . . . .	88
5.9.5.20 ID . . . . .	88
5.9.5.21 lgcTox . . . . .	88
5.9.5.22 lgcToy . . . . .	89
5.9.5.23 lgcToz . . . . .	89
5.9.5.24 Llx . . . . .	89
5.9.5.25 Lly . . . . .	89

5.9.5.26 Llz	90
5.9.5.27 lx	90
5.9.5.28 ly	90
5.9.5.29 lz	90
5.9.5.30 nx	91
5.9.5.31 ny	91
5.9.5.32 nz	91
5.9.5.33 rgcTox	91
5.9.5.34 rgcToy	92
5.9.5.35 rgcToz	92
5.9.5.36 statusFlags	92
5.9.5.37 u	92
5.9.5.38 uAux	93
5.9.5.39 uAuxData	93
5.9.5.40 uData	93
5.9.5.41 uLocal	93
5.9.5.42 uTox	94
5.9.5.43 uToy	94
5.9.5.44 uToz	94
5.9.5.45 x0	94
5.9.5.46 xTou	95
5.9.5.47 y0	95
5.9.5.48 yTou	95
5.9.5.49 z0	95
5.9.5.50 zTou	96
5.10 OutputManager Class Reference	96
5.10.1 Detailed Description	97
5.10.2 Constructor & Destructor Documentation	97
5.10.2.1 OutputManager()	97
5.10.3 Member Function Documentation	97
5.10.3.1 generateOutputFolder()	98
5.10.3.2 getSimCode()	99
5.10.3.3 outUState()	99
5.10.3.4 set_outputStyle()	101
5.10.3.5 SimCodeGenerator()	101
5.10.4 Field Documentation	102
5.10.4.1 outputStyle	102
5.10.4.2 Path	102
5.10.4.3 simCode	103
5.11 PlaneWave Class Reference	103
5.11.1 Detailed Description	104
5.11.2 Field Documentation	104

5.11.2.1 kx . . . . .	104
5.11.2.2 ky . . . . .	104
5.11.2.3 kz . . . . .	104
5.11.2.4 phix . . . . .	105
5.11.2.5 phiy . . . . .	105
5.11.2.6 phiz . . . . .	105
5.11.2.7 px . . . . .	105
5.11.2.8 py . . . . .	106
5.11.2.9 pz . . . . .	106
5.12 planewave Struct Reference . . . . .	106
5.12.1 Detailed Description . . . . .	106
5.12.2 Field Documentation . . . . .	107
5.12.2.1 k . . . . .	107
5.12.2.2 p . . . . .	107
5.12.2.3 phi . . . . .	107
5.13 PlaneWave1D Class Reference . . . . .	108
5.13.1 Detailed Description . . . . .	108
5.13.2 Constructor & Destructor Documentation . . . . .	109
5.13.2.1 PlaneWave1D() . . . . .	109
5.13.3 Member Function Documentation . . . . .	109
5.13.3.1 addToSpace() . . . . .	110
5.14 PlaneWave2D Class Reference . . . . .	110
5.14.1 Detailed Description . . . . .	111
5.14.2 Constructor & Destructor Documentation . . . . .	111
5.14.2.1 PlaneWave2D() . . . . .	111
5.14.3 Member Function Documentation . . . . .	112
5.14.3.1 addToSpace() . . . . .	112
5.15 PlaneWave3D Class Reference . . . . .	113
5.15.1 Detailed Description . . . . .	114
5.15.2 Constructor & Destructor Documentation . . . . .	114
5.15.2.1 PlaneWave3D() . . . . .	114
5.15.3 Member Function Documentation . . . . .	115
5.15.3.1 addToSpace() . . . . .	115
5.16 Simulation Class Reference . . . . .	115
5.16.1 Detailed Description . . . . .	117
5.16.2 Constructor & Destructor Documentation . . . . .	117
5.16.2.1 Simulation() . . . . .	118
5.16.2.2 ~Simulation() . . . . .	118
5.16.3 Member Function Documentation . . . . .	119
5.16.3.1 addInitialConditions() . . . . .	119
5.16.3.2 addPeriodicCLayerInX() . . . . .	120
5.16.3.3 addPeriodicCLayerInXY() . . . . .	121

5.16.3.4 advanceToTime() . . . . .	122
5.16.3.5 checkFlag() . . . . .	123
5.16.3.6 checkNoFlag() . . . . .	124
5.16.3.7 get_cart_comm() . . . . .	125
5.16.3.8 initializeCVODEobject() . . . . .	125
5.16.3.9 initializePatchwork() . . . . .	127
5.16.3.10 outAllFieldData() . . . . .	128
5.16.3.11 setDiscreteDimensionsOfLattice() . . . . .	129
5.16.3.12 setInitialConditions() . . . . .	130
5.16.3.13 setPhysicalDimensionsOfLattice() . . . . .	131
5.16.3.14 start() . . . . .	132
5.16.4 Field Documentation . . . . .	133
5.16.4.1 cvode_mem . . . . .	133
5.16.4.2 icsettings . . . . .	134
5.16.4.3 lattice . . . . .	134
5.16.4.4 latticePatch . . . . .	134
5.16.4.5 NLS . . . . .	134
5.16.4.6 outputManager . . . . .	135
5.16.4.7 statusFlags . . . . .	135
5.16.4.8 t . . . . .	135
5.17 TimeEvolution Class Reference . . . . .	135
5.17.1 Detailed Description . . . . .	136
5.17.2 Member Function Documentation . . . . .	136
5.17.2.1 f() . . . . .	136
5.17.3 Field Documentation . . . . .	137
5.17.3.1 c . . . . .	137
5.17.3.2 TimeEvolver . . . . .	137
<b>6 File Documentation</b> . . . . .	<b>139</b>
6.1 README.md File Reference . . . . .	139
6.2 src/DerivationStencils.cpp File Reference . . . . .	139
6.2.1 Detailed Description . . . . .	139
6.3 DerivationStencils.cpp . . . . .	140
6.4 src/DerivationStencils.h File Reference . . . . .	140
6.4.1 Detailed Description . . . . .	142
6.4.2 Function Documentation . . . . .	142
6.4.2.1 s10b() [1/2] . . . . .	142
6.4.2.2 s10b() [2/2] . . . . .	143
6.4.2.3 s10c() [1/2] . . . . .	143
6.4.2.4 s10c() [2/2] . . . . .	144
6.4.2.5 s10f() [1/2] . . . . .	144
6.4.2.6 s10f() [2/2] . . . . .	145

---

6.4.2.7 <code>s11b()</code> [1/2] . . . . .	145
6.4.2.8 <code>s11b()</code> [2/2] . . . . .	146
6.4.2.9 <code>s11f()</code> [1/2] . . . . .	146
6.4.2.10 <code>s11f()</code> [2/2] . . . . .	147
6.4.2.11 <code>s12b()</code> [1/2] . . . . .	147
6.4.2.12 <code>s12b()</code> [2/2] . . . . .	148
6.4.2.13 <code>s12c()</code> [1/2] . . . . .	148
6.4.2.14 <code>s12c()</code> [2/2] . . . . .	149
6.4.2.15 <code>s12f()</code> [1/2] . . . . .	149
6.4.2.16 <code>s12f()</code> [2/2] . . . . .	150
6.4.2.17 <code>s13b()</code> [1/2] . . . . .	150
6.4.2.18 <code>s13b()</code> [2/2] . . . . .	151
6.4.2.19 <code>s13f()</code> [1/2] . . . . .	151
6.4.2.20 <code>s13f()</code> [2/2] . . . . .	152
6.4.2.21 <code>s1b()</code> [1/2] . . . . .	152
6.4.2.22 <code>s1b()</code> [2/2] . . . . .	153
6.4.2.23 <code>s1f()</code> [1/2] . . . . .	153
6.4.2.24 <code>s1f()</code> [2/2] . . . . .	154
6.4.2.25 <code>s2b()</code> [1/2] . . . . .	154
6.4.2.26 <code>s2b()</code> [2/2] . . . . .	155
6.4.2.27 <code>s2c()</code> [1/2] . . . . .	155
6.4.2.28 <code>s2c()</code> [2/2] . . . . .	156
6.4.2.29 <code>s2f()</code> [1/2] . . . . .	156
6.4.2.30 <code>s2f()</code> [2/2] . . . . .	157
6.4.2.31 <code>s3b()</code> [1/2] . . . . .	157
6.4.2.32 <code>s3b()</code> [2/2] . . . . .	158
6.4.2.33 <code>s3f()</code> [1/2] . . . . .	158
6.4.2.34 <code>s3f()</code> [2/2] . . . . .	159
6.4.2.35 <code>s4b()</code> [1/2] . . . . .	159
6.4.2.36 <code>s4b()</code> [2/2] . . . . .	160
6.4.2.37 <code>s4c()</code> [1/2] . . . . .	160
6.4.2.38 <code>s4c()</code> [2/2] . . . . .	161
6.4.2.39 <code>s4f()</code> [1/2] . . . . .	161
6.4.2.40 <code>s4f()</code> [2/2] . . . . .	162
6.4.2.41 <code>s5b()</code> [1/2] . . . . .	162
6.4.2.42 <code>s5b()</code> [2/2] . . . . .	163
6.4.2.43 <code>s5f()</code> [1/2] . . . . .	163
6.4.2.44 <code>s5f()</code> [2/2] . . . . .	164
6.4.2.45 <code>s6b()</code> [1/2] . . . . .	164
6.4.2.46 <code>s6b()</code> [2/2] . . . . .	165
6.4.2.47 <code>s6c()</code> [1/2] . . . . .	165
6.4.2.48 <code>s6c()</code> [2/2] . . . . .	166

---

6.4.2.49 <code>s6f()</code> [1/2] . . . . .	166
6.4.2.50 <code>s6f()</code> [2/2] . . . . .	167
6.4.2.51 <code>s7b()</code> [1/2] . . . . .	167
6.4.2.52 <code>s7b()</code> [2/2] . . . . .	168
6.4.2.53 <code>s7f()</code> [1/2] . . . . .	168
6.4.2.54 <code>s7f()</code> [2/2] . . . . .	169
6.4.2.55 <code>s8b()</code> [1/2] . . . . .	169
6.4.2.56 <code>s8b()</code> [2/2] . . . . .	170
6.4.2.57 <code>s8c()</code> [1/2] . . . . .	170
6.4.2.58 <code>s8c()</code> [2/2] . . . . .	171
6.4.2.59 <code>s8f()</code> [1/2] . . . . .	171
6.4.2.60 <code>s8f()</code> [2/2] . . . . .	172
6.4.2.61 <code>s9b()</code> [1/2] . . . . .	172
6.4.2.62 <code>s9b()</code> [2/2] . . . . .	173
6.4.2.63 <code>s9f()</code> [1/2] . . . . .	173
6.4.2.64 <code>s9f()</code> [2/2] . . . . .	174
6.5 <code>DerivationStencils.h</code> . . . . .	174
6.6 <code>src/ICSetters.cpp</code> File Reference . . . . .	178
6.6.1 Detailed Description . . . . .	178
6.7 <code>ICSetters.cpp</code> . . . . .	178
6.8 <code>src/ICSetters.h</code> File Reference . . . . .	183
6.8.1 Detailed Description . . . . .	184
6.9 <code>ICSetters.h</code> . . . . .	184
6.10 <code>src/LatticePatch.cpp</code> File Reference . . . . .	187
6.10.1 Detailed Description . . . . .	188
6.10.2 Function Documentation . . . . .	188
6.10.2.1 <code>check_error()</code> . . . . .	188
6.10.2.2 <code>check_retval()</code> . . . . .	189
6.10.2.3 <code>errorKill()</code> . . . . .	190
6.10.2.4 <code>generatePatchwork()</code> . . . . .	191
6.11 <code>LatticePatch.cpp</code> . . . . .	192
6.12 <code>src/LatticePatch.h</code> File Reference . . . . .	203
6.12.1 Detailed Description . . . . .	205
6.12.2 Function Documentation . . . . .	205
6.12.2.1 <code>check_error()</code> . . . . .	205
6.12.2.2 <code>check_retval()</code> . . . . .	206
6.12.2.3 <code>errorKill()</code> . . . . .	207
6.12.3 Variable Documentation . . . . .	207
6.12.3.1 <code>BuffersInitialized</code> . . . . .	208
6.12.3.2 <code>FLatticeDimensionSet</code> . . . . .	208
6.12.3.3 <code>FLatticePatchSetUp</code> . . . . .	208
6.12.3.4 <code>GhostLayersInitialized</code> . . . . .	208

---

6.12.3.5 TranslocationLookupSetUp . . . . .	209
6.13 LatticePatch.h . . . . .	209
6.14 src/main.cpp File Reference . . . . .	212
6.14.1 Detailed Description . . . . .	212
6.14.2 Function Documentation . . . . .	212
6.14.2.1 main() . . . . .	213
6.15 main.cpp . . . . .	220
6.16 src/Outputters.cpp File Reference . . . . .	224
6.16.1 Detailed Description . . . . .	224
6.17 Outputters.cpp . . . . .	224
6.18 src/Outputters.h File Reference . . . . .	226
6.18.1 Detailed Description . . . . .	227
6.19 Outputters.h . . . . .	227
6.20 src/SimulationClass.cpp File Reference . . . . .	228
6.20.1 Detailed Description . . . . .	228
6.21 SimulationClass.cpp . . . . .	228
6.22 src/SimulationClass.h File Reference . . . . .	232
6.22.1 Detailed Description . . . . .	233
6.22.2 Variable Documentation . . . . .	233
6.22.2.1 CvodeObjectSetUp . . . . .	234
6.22.2.2 LatticeDiscreteSetUp . . . . .	234
6.22.2.3 LatticePatchworkSetUp . . . . .	234
6.22.2.4 LatticePhysicalSetUp . . . . .	234
6.22.2.5 SimulationStarted . . . . .	235
6.23 SimulationClass.h . . . . .	235
6.24 src/SimulationFunctions.cpp File Reference . . . . .	236
6.24.1 Detailed Description . . . . .	237
6.24.2 Function Documentation . . . . .	237
6.24.2.1 Sim1D() . . . . .	237
6.24.2.2 Sim2D() . . . . .	240
6.24.2.3 Sim3D() . . . . .	243
6.24.2.4 timer() . . . . .	246
6.25 SimulationFunctions.cpp . . . . .	247
6.26 src/SimulationFunctions.h File Reference . . . . .	250
6.26.1 Detailed Description . . . . .	251
6.26.2 Function Documentation . . . . .	251
6.26.2.1 Sim1D() . . . . .	251
6.26.2.2 Sim2D() . . . . .	254
6.26.2.3 Sim3D() . . . . .	257
6.26.2.4 timer() . . . . .	260
6.27 SimulationFunctions.h . . . . .	261
6.28 src/TimeEvolutionFunctions.cpp File Reference . . . . .	262

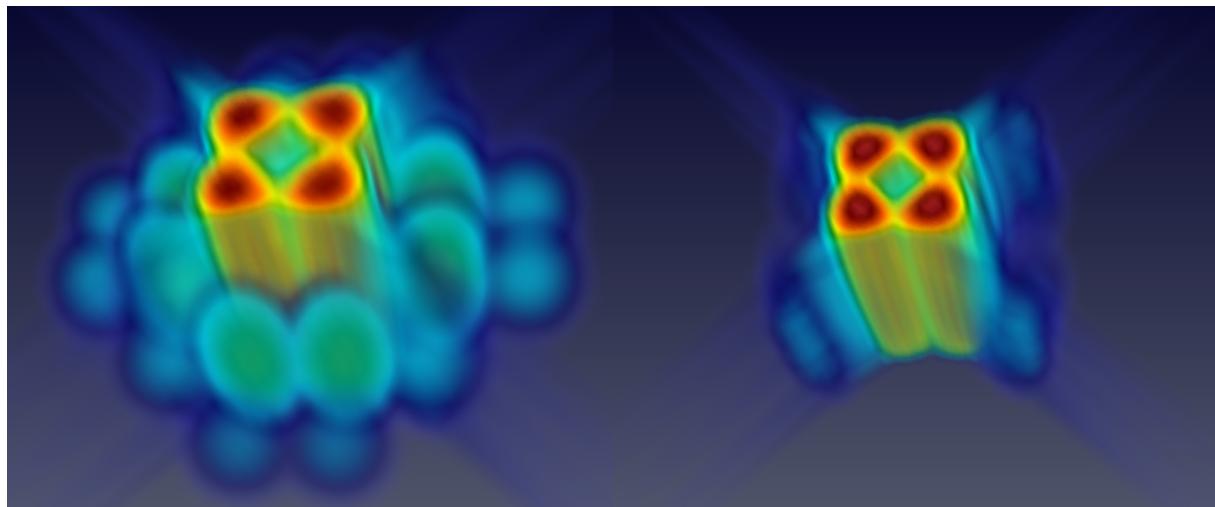
6.28.1 Detailed Description . . . . .	262
6.28.2 Function Documentation . . . . .	262
6.28.2.1 linear1DProp() . . . . .	263
6.28.2.2 linear2DProp() . . . . .	264
6.28.2.3 linear3DProp() . . . . .	266
6.28.2.4 nonlinear1DProp() . . . . .	268
6.28.2.5 nonlinear2DProp() . . . . .	273
6.28.2.6 nonlinear3DProp() . . . . .	277
6.29 TimeEvolutionFunctions.cpp . . . . .	281
6.30 src/TimeEvolutionFunctions.h File Reference . . . . .	289
6.30.1 Detailed Description . . . . .	291
6.30.2 Function Documentation . . . . .	291
6.30.2.1 linear1DProp() . . . . .	291
6.30.2.2 linear2DProp() . . . . .	292
6.30.2.3 linear3DProp() . . . . .	294
6.30.2.4 nonlinear1DProp() . . . . .	296
6.30.2.5 nonlinear2DProp() . . . . .	301
6.30.2.6 nonlinear3DProp() . . . . .	305
6.31 TimeEvolutionFunctions.h . . . . .	309
<b>Index</b>	<b>311</b>

## Chapter 1

# HEWES: Heisenberg-Euler Weak-Field Expansion Simulator

The Heisenberg-Euler Weak-Field Expansion Simulator is a solver for the all-optical QED vacuum. It solves the equations of motion for electromagnetic waves in the Heisenberg-Euler effective QED theory in the weak-field expansion with up to six-photon processes.

There is a [paper](#) that introduces the algorithm and shows remarkable results and a [Mendeley Data repository](#) with extra and supplementary materials.



**Figure 1.1 Harmonic Generation 3D**

### 1.1 Contents

- Preparing the Makefile
- Short User Manual
  - Hints for Settings
  - Note on Resource Occupation
  - Note on Output Analysis
- Authors

## 1.2 Preparing the Makefile

The following descriptions assume you are using a Unix-like system.

The `make` utility is used for building and a recent compiler version is required. Features up to the C++20 standard are used. `OpenMP` is optional to enforce more vectorization and enable multi-threading. The latter is useful for performance only when a very large number of nodes is used.

Additionally required software:

- An MPI implementation such as `OpenMPI` or `MPICH`.
- The `SUNDIALS` package or the `CVODE` solver.

Version 6 is required. The code is presumably compliant with the upcoming version 7.

For the installation of `SUNDIALS`, `CMake` is required. Enable MPI and specify the directory of the `mpicxx` wrapper for use of the MPI-based `NVECTOR_PARALLEL` module. Make sure to edit the `SUNDIALS` include and library directories in the provided minimal `Makefile`.

## 1.3 Short User Manual

You have full control over all high-level simulation settings via the `main.cpp` file.

- First, specify the path you want the output data to go via the variable `outputDirectory`.
- Second, decide if you want to simulate in 1D, 2D, or 3D and uncomment only that full section.  
You can then specify
  - the relative and absolute integration tolerances of the `CVODE` solver.  
Recommended values are between 1e-12 and 1e-16.
  - the order of accuracy of the numerical scheme via the stencil order.  
You can choose an integer in the range 1-13.
  - the physical side lengths of the grid in meters.
  - the number of lattice points per dimension.
  - the slicing of the lattice into patches (only for 2D and 3D simulations, automatic in 1D) – this determines the number of patches and therefore the required distinct processing units for MPI.  
The total number of processes is given by the product of patches in any dimension.  
Note: In the 3D case patches are required to be cubic in terms of lattice points. This is decisive for computational efficiency and checked at compile-time.
  - whether to have periodic or vanishing boundary values (currently has to be chosen periodic).
  - whether you want to simulate on top of the linear vacuum only 4-photon processes (1), 6-photon processes (2), both (3), or none (0) – the linear Maxwell case.
  - the total time of the simulation in units  $c=1$ , i.e., the distance propagated by the light waves in meters.
  - the number of time steps that will be solved stepwise by `CVODE`.  
In order to keep interpolation errors small do not choose this number too small.
  - the multiple of steps at which you want the data to be written to disk.
  - the output format. It can be 'c' for comma separated values (csv), or 'b' for binary. For csv format the name of the files written to the output directory is of the form `{step_number}_{process_↔ number}.csv`. For binary output all data per step are written into one file and the step number is the name of the file.

- which electromagnetic waveform(s) you want to propagate.

You can choose between a plane wave (not much physical content, but useful for checks) and implementations of Gaussians in 1D, 2D, and 3D. Their parameters can be tuned.

A description of the wave implementations is given in [ref.pdf](#). Note that the 3D Gaussians, as they are implemented up to now, should be propagated in the xy-plane. More waveform implementations will follow in subsequent versions of the code.

A doxygen-generated complete code reference is provided with [ref.pdf](#).

- Third, in the `[src](src)` directory, build the executable [Simulation](#) via the `make` command.

- Forth, run the simulation.

Make sure to use `src` as working directory as the code uses a relative path to log the configuration in [main.cpp](#).

You determine the number of processes via the MPI execution command. Note that in 2D and 3D simulations this number has to coincide with the actual number of patches, as described above.

Here, the simulation would be executed distributed over four processes:

```
mpirun -np 4 ./Simulation
```

- Monitor `stdout` and `stderr`. The unique simulation identifier number (starting timestep = name of data directory), the process steps, and the used wall times per step are printed on `stdout`. Errors are printed on `stderr`.

**Note:** Convergence of the employed *CVODE* solver cannot be guaranteed and issues of this kind can hardly be predicted. On top, they are even system dependent. Piece of advice: Only pass decimal numbers for the grid settings and initial conditions.

*CVODE* warnings and errors are reported on `stdout` and `stderr`.

A `config.txt` file containing the configuration part of [main.cpp](#) is written to the output directory in order to log the simulation settings of each particular run.

You can remove the object files and the executable via `make clean`.

### 1.3.1 Note on Simulation Settings

You may want to start with two Gaussian pulses in 1D colliding head-on in a pump-probe setup. For this event, specify a high-frequency probe pulse with a low amplitude and a low-frequency pump pulse with a high frequency. Both frequencies should be chosen to be below a forth of the Nyquist frequency to minimize nonphysical dispersion effects on the lattice. The wavelengths should neither be chosen too large (bulky wave) on a fine patchwork of narrow patches. Their communication might be problematic with too small halo layer depths. You would observe a blurring over time. The amplitudes need be below 1 – the critical field strength – for the weak-field expansion to be valid.

You can then investigate the arising of higher harmonics in frequency space via a Fourier analysis. The signals from the higher harmonics can be highlighted by subtracting the results of the same simulation in the linear Maxwell vacuum. You will be left with the nonlinear effects.

Choosing the probe pulse to be polarized with an angle to the polarization of the pump you may observe a fractional polarization flip of the probe due to their nonlinear interaction.

Decide beforehand which steps you need to be written to disk for your analysis.

Example scenarios of colliding Gaussians are preconfigured for any dimension.

### 1.3.2 Note on Resource Occupation

The computational load depends mostly on the grid size and resolution. The order of accuracy of the numerical scheme and *CVODE* are rather secondary except for simulations running on many processing units, as the communication load is dependent on the stencil order.

Simulations in 1D are relatively cheap and can easily be run on a modern laptop within seconds. The output size per step is usually less than a megabyte.

Simulations in 2D with about one million grid points are still feasible for a personal machine but might take a couple of minutes to finish. The output size per step is in the range of some dozen megabytes.

Sensible simulations in 3D require large memory resources and therefore need to be run on distributed systems. Hundreds of cores can be kept busy for many hours or days. The output size quickly amounts to dozens of gigabytes for just a single state.

### 1.3.3 Note on Output Analysis

The field data are either written in csv format to one file per MPI process, the ending of which (after an underscore) corresponds to the process number, as described above. This is the simplest solution for smaller simulations and a portable way that also works fast and is straightforward to analyze.

Or, the option strictly recommended for larger write operations, in binary format with a single file per output step. Raw bytes are written to the files as they are in memory. This option is more performant and achieved with MPI IO. However, there is no guarantee of portability; postprocessing/conversion is required. The step number is the file name.

A `SimResults` folder is created in the chosen output directory if it does not exist and therein a folder named after the starting timestep of the simulation (in the form `yy-mm-dd_hh-MM-ss`) is created. This is where the output files are written into.

There are six columns in the csv files, corresponding to the six components of the electromagnetic field: `$E_x$, $E_y$, $E_z$, $B_x$, $B_y$, $B_z$. Each row corresponds to one lattice point.`

Postprocessing is required to read-in the files in order. A [Python module](#) taking care of this is provided.

Likewise, [another Python module](#) is provided to read the binary data of a selected field component into a numpy array – its portability, however, cannot be guaranteed.

The process numbers first align along dimension 1 until the number of patches is that direction is reached, then continue on dimension two and finally fill dimension 3. For example, for a 3D simulation on  $4 \times 4 \times 4 = 64$  cores, the field data is divided over the patches as follows:

$z=1$	$z=2$	$z=3$	$z=4$
$x$	$x$	$x$	$x$
$\wedge$	$\wedge$	$\wedge$	$\wedge$
1   0 4 8 12	1   16 20 24 28	1   16 20 24 28	1   16 20 24 28
2   1 5 9 13	2   17 21 25 29	2   17 21 25 29	2   17 21 25 29
3   2 6 10 14	3   18 22 26 30	3   18 22 26 30	3   18 22 26 30
4   3 7 11 15	4   19 23 27 31	4   19 23 27 31	4   19 23 27 31
----->	----->	----->	----->
1 2 3 4 y	1 2 3 4 y	1 2 3 4 y	1 2 3 4 y

The axes denote the physical dimensions that are each divided into 4 sectors in this example. The numbers inside the  $4 \times 4$  squares indicate the process number, which is the number of the patch and also the number at the end of the corresponding output csv file. The ordering of the array within a patch follows the standard C convention and can be reshaped in 2D and 3D to the actual size of the path.

More information describing settings and analysis procedures used for actual scientific results are given in an open-access [paper](#) and a collection of corresponding analysis notebooks are uploaded to a [Mendeley Data repository](#). Some example Python analysis scripts can be found in the [examples](examples). The [first steps](#) demonstrate how the simulated data is accurately read-in from disk to numpy arrays using the provided [get field data module](#). [Harmonic generation](#) in various forms is sketched as one application showing nonlinear quantum vacuum effects. Analyses of 3D simulations are more involved due to large volumes of data. Visualization requires tools like Paraview; examples are shown [here](#). There is however *no simulation data provided* as it would make the repository size unnecessarily large.

## 1.4 Authors

- Arnau Pons Domenech
- Hartmut Ruhl ( [hartmut.ruhl@physik.uni-muenchen.de](mailto:hartmut.ruhl@physik.uni-muenchen.de))
- Andreas Lindner ( [and.lindner@physik.uni-muenchen.de](mailto:and.lindner@physik.uni-muenchen.de))
- Baris Ölmez ( [b.oelmez@physik.uni-muenchen.de](mailto:b.oelmez@physik.uni-muenchen.de))

# Chapter 2

## Hierarchical Index

### 2.1 Class Hierarchy

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

Gauss1D . . . . .	11
Gauss2D . . . . .	17
Gauss3D . . . . .	23
gaussian1D . . . . .	28
gaussian2D . . . . .	30
gaussian3D . . . . .	32
ICSetter . . . . .	34
Lattice . . . . .	43
LatticePatch . . . . .	57
OutputManager . . . . .	96
PlaneWave . . . . .	103
PlaneWave1D . . . . .	108
PlaneWave2D . . . . .	110
PlaneWave3D . . . . .	113
planewave . . . . .	106
Simulation . . . . .	115
TimeEvolution . . . . .	135



# Chapter 3

## Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">Gauss1D</a>	Class for Gaussian pulses in 1D . . . . .	11
<a href="#">Gauss2D</a>	Class for Gaussian pulses in 2D . . . . .	17
<a href="#">Gauss3D</a>	Class for Gaussian pulses in 3D . . . . .	23
<a href="#">gaussian1D</a>	1D Gaussian wave structure . . . . .	28
<a href="#">gaussian2D</a>	2D Gaussian wave structure . . . . .	30
<a href="#">gaussian3D</a>	3D Gaussian wave structure . . . . .	32
<a href="#">ICSetter</a>	<a href="#">ICSetter</a> class to initialize wave types with default parameters . . . . .	34
<a href="#">Lattice</a>	<a href="#">Lattice</a> class for the construction of the enveloping discrete simulation space . . . . .	43
<a href="#">LatticePatch</a>	<a href="#">LatticePatch</a> class for the construction of the patches in the enveloping lattice . . . . .	57
<a href="#">OutputManager</a>	Output Manager class to generate and coordinate output writing to disk . . . . .	96
<a href="#">PlaneWave</a>	Super-class for plane waves . . . . .	103
<a href="#">planewave</a>	Plane wave structure . . . . .	106
<a href="#">PlaneWave1D</a>	Class for plane waves in 1D . . . . .	108
<a href="#">PlaneWave2D</a>	Class for plane waves in 2D . . . . .	110
<a href="#">PlaneWave3D</a>	Class for plane waves in 3D . . . . .	113
<a href="#">Simulation</a>	Simulation class to instantiate the whole walkthrough of a <a href="#">Simulation</a> . . . . .	115
<a href="#">TimeEvolution</a>	Monostate <a href="#">TimeEvolution</a> class to propagate the field data in time in a given order of the HE weak-field expansion . . . . .	135



# Chapter 4

## File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/DerivationStencils.cpp	
Empty. All definitions in the header . . . . .	139
src/DerivationStencils.h	
Definition of derivation stencils from order 1 to 13 . . . . .	140
src/ICSetters.cpp	
Implementation of the plane wave and Gaussian wave packets . . . . .	178
src/ICSetters.h	
Declaration of the plane wave and Gaussian wave packets . . . . .	183
src/LatticePatch.cpp	
Construction of the overall envelope lattice and the lattice patches . . . . .	187
src/LatticePatch.h	
Declaration of the lattice and lattice patches . . . . .	203
src/main.cpp	
Main function to configure the user's simulation settings . . . . .	212
src/Outputters.cpp	
Generation of output writing to disk . . . . .	224
src/Outputters.h	
OutputManager class to outstream simulation data . . . . .	226
src/SimulationClass.cpp	
Interface to the whole Simulation procedure: from wave settings over lattice construction, time evolution and outputs (also all relevant CVODE steps are performed here) . . . . .	228
src/SimulationClass.h	
Class for the Simulation object calling all functionality: from wave settings over lattice construction, time evolution and outputs initialization of the CVode object . . . . .	232
src/SimulationFunctions.cpp	
Implementation of the complete simulation functions for 1D, 2D, and 3D, as called in the main function . . . . .	236
src/SimulationFunctions.h	
Full simulation functions for 1D, 2D, and 3D used in main.cpp . . . . .	250
src/TimeEvolutionFunctions.cpp	
Implementation of functions to propagate data vectors in time according to Maxwell's equations, and various orders in the HE weak-field expansion . . . . .	262
src/TimeEvolutionFunctions.h	
Functions to propagate data vectors in time according to Maxwell's equations, and various orders in the HE weak-field expansion . . . . .	289



# Chapter 5

# Data Structure Documentation

## 5.1 Gauss1D Class Reference

class for Gaussian pulses in 1D

```
#include <src/ICSetters.h>
```

### Public Member Functions

- `Gauss1D` (std::array< surrealtype, 3 > k={1, 0, 0}, std::array< surrealtype, 3 > p={0, 0, 1}, std::array< surrealtype, 3 > xo={0, 0, 0}, surrealtype phig\_=1.0, std::array< surrealtype, 3 > phi={0, 0, 0})  
*construction with default parameters*
- void `addToSpace` (surrealtype x, surrealtype y, surrealtype z, surrealtype \*pTo6Space) const  
*function for the actual implementation in space*

### Private Attributes

- surrealtype `kx`  
*wavenumber  $k_x$*
- surrealtype `ky`  
*wavenumber  $k_y$*
- surrealtype `kz`  
*wavenumber  $k_z$*
- surrealtype `px`  
*polarization & amplitude in x-direction,  $p_x$*
- surrealtype `py`  
*polarization & amplitude in y-direction,  $p_y$*
- surrealtype `pz`  
*polarization & amplitude in z-direction,  $p_z$*
- surrealtype `phix`  
*phase shift in x-direction,  $\phi_x$*
- surrealtype `phiy`  
*phase shift in y-direction,  $\phi_y$*
- surrealtype `phiz`

- sunrealtype `x0z`  
*center of pulse in z-direction,  $\phi_z$*
- sunrealtype `x0x`  
*center of pulse in x-direction,  $x_0$*
- sunrealtype `x0y`  
*center of pulse in y-direction,  $y_0$*
- sunrealtype `x0z`  
*center of pulse in z-direction,  $z_0$*
- sunrealtype `phig`  
*pulse width  $\Phi_g$*

### 5.1.1 Detailed Description

class for Gaussian pulses in 1D

They are given in the form  $\vec{E} = \vec{p} \exp\left(-(\vec{x} - \vec{x}_0)^2/\Phi_g^2\right) \cos(\vec{k} \cdot \vec{x})$

Definition at line 83 of file [ICSetters.h](#).

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 Gauss1D()

```
Gauss1D::Gauss1D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > xo = {0, 0, 0},
    sunrealtype phig_ = 1.0,
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

construction with default parameters

[Gauss1D](#) construction with

- wavevectors  $k_x$
- $k_y$
- $k_z$  normalized to  $1/\lambda$
- amplitude (polarization) in x-direction
- amplitude (polarization) in y-direction
- amplitude (polarization) in z-direction
- phase shift in x-direction
- phase shift in y-direction
- phase shift in z-direction
- width

- shift from origin in x-direction
- shift from origin in y-direction
- shift from origin in z-direction

Definition at line 125 of file [ICSetters.cpp](#).

```
00127
00128   kx = k[0];    /** - wavevectors \f$ k_x \f$ */
00129   ky = k[1];    /** - \f$ k_y \f$ */
00130   kz = k[2];    /** - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00131   px = p[0];    /** - amplitude (polarization) in x-direction */
00132   py = p[1];    /** - amplitude (polarization) in y-direction */
00133   pz = p[2];    /** - amplitude (polarization) in z-direction */
00134   phix = phi[0]; /* - phase shift in x-direction */
00135   phiy = phi[1]; /* - phase shift in y-direction */
00136   phiz = phi[2]; /* - phase shift in z-direction */
00137   phig = phig_; /* - width */
00138   x0x = xo[0];  /* - shift from origin in x-direction*/
00139   x0y = xo[1];  /* - shift from origin in y-direction*/
00140   x0z = xo[2];  /* - shift from origin in z-direction*/
00141 }
```

References [kx](#), [ky](#), [kz](#), [phig](#), [phix](#), [phiy](#), [phiz](#), [px](#), [py](#), [pz](#), [x0x](#), [x0y](#), and [x0z](#).

### 5.1.3 Member Function Documentation

#### 5.1.3.1 addToSpace()

```
void Gauss1D::addToSpace (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space ) const
```

function for the actual implementation in space

[Gauss1D](#) implementation in space

Definition at line 144 of file [ICSetters.cpp](#).

```
00145
00146   const sunrealtype wavelength =
00147     sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00148   x = x - x0x; /* x-coordinate minus shift from origin */
00149   y = y - x0y; /* y-coordinate minus shift from origin */
00150   z = z - x0z; /* z-coordinate minus shift from origin */
00151   const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00152     std::numbers::pi; /* \f$ 2\pi \f$ */
00153   const sunrealtype envelopeAmp =
00154     exp(-(x * x + y * y + z * z) / phig / phig); /* enveloping Gauss shape */
00155   // Gaussian wave definition
00156   const std::array<sunrealtype, 3> E{                                /* E-field vector */
00157     px * cos(kScalarX - phix) * envelopeAmp, /* \f$ E_x \f$ */
00158     py * cos(kScalarX - phiy) * envelopeAmp, /* \f$ E_y \f$ */
00159     pz * cos(kScalarX - phiz) * envelopeAmp}; /* \f$ E_z \f$ */
00160   // Put E-field into space
00161   pTo6Space[0] += E[0];
00162   pTo6Space[1] += E[1];
00163   pTo6Space[2] += E[2];
00164   // and B-field
00165   pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00166   pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00167   pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00168
00169 }
```

References [kx](#), [ky](#), [kz](#), [phig](#), [phix](#), [phiy](#), [phiz](#), [px](#), [py](#), [pz](#), [x0x](#), [x0y](#), and [x0z](#).

## 5.1.4 Field Documentation

### 5.1.4.1 kx

```
sunrealtype Gauss1D::kx [private]
```

wavenumber  $k_x$

Definition at line 86 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.2 ky

```
sunrealtype Gauss1D::ky [private]
```

wavenumber  $k_y$

Definition at line 88 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.3 kz

```
sunrealtype Gauss1D::kz [private]
```

wavenumber  $k_z$

Definition at line 90 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.4 phig

```
sunrealtype Gauss1D::phig [private]
```

pulse width  $\Phi_g$

Definition at line 110 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

#### 5.1.4.5 phix

```
sunrealtype Gauss1D::phix [private]
```

phase shift in x-direction,  $\phi_x$

Definition at line 98 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

#### 5.1.4.6 phiy

```
sunrealtype Gauss1D::phiy [private]
```

phase shift in y-direction,  $\phi_y$

Definition at line 100 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

#### 5.1.4.7 phiz

```
sunrealtype Gauss1D::phiz [private]
```

phase shift in z-direction,  $\phi_z$

Definition at line 102 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

#### 5.1.4.8 px

```
sunrealtype Gauss1D::px [private]
```

polarization & amplitude in x-direction,  $p_x$

Definition at line 92 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.9 **py**

`sunrealtype Gauss1D::py [private]`

polarization & amplitude in y-direction,  $p_y$

Definition at line 94 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.10 **pz**

`sunrealtype Gauss1D::pz [private]`

polarization & amplitude in z-direction,  $p_z$

Definition at line 96 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.11 **x0x**

`sunrealtype Gauss1D::x0x [private]`

center of pulse in x-direction,  $x_0$

Definition at line 104 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.12 **x0y**

`sunrealtype Gauss1D::x0y [private]`

center of pulse in y-direction,  $y_0$

Definition at line 106 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

### 5.1.4.13 x0z

`sunrealtype Gauss1D::x0z [private]`

center of pulse in z-direction,  $z_0$

Definition at line 108 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss1D\(\)](#).

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

- [src/ICSetters.h](#)
- [src/ICSetters.cpp](#)

## 5.2 Gauss2D Class Reference

class for Gaussian pulses in 2D

```
#include <src/ICSetters.h>
```

### Public Member Functions

- [Gauss2D](#) (`std::array< unrealtype, 3 > dis_{0, 0, 0}, std::array< unrealtype, 3 > axis_{1, 0, 0}, unrealtype Amp_=1.0, unrealtype phip_=0, unrealtype w0_=1e-5, unrealtype zr_=4e-5, unrealtype Ph0_=2e-5, unrealtype PhA_=0.45e-5}`)  
*construction with default parameters*
- void [addToSpace](#) (`unrealtype x, unrealtype y, unrealtype z, unrealtype *pTo6Space`) const  
*function for the actual implementation in space*

### Private Attributes

- `std::array< unrealtype, 3 > dis`  
*distance maximum to origin*
- `std::array< unrealtype, 3 > axis`  
*normalized propagation axis*
- `unrealtype Amp`  
*amplitude A*
- `unrealtype phip`  
*polarization rotation from TE-mode around propagation direction*
- `unrealtype w0`  
*taille  $\omega_0$*
- `unrealtype zr`  
*Rayleigh length  $z_R = \pi\omega_0^2/\lambda$*
- `unrealtype Ph0`  
*center of beam  $\Phi_0$*
- `unrealtype PhA`  
*length of beam  $\Phi_A$*
- `unrealtype A1`  
*amplitude projection on TE-mode*
- `unrealtype A2`  
*amplitude projection on xy-plane*
- `unrealtype lambda`  
*wavelength  $\lambda$*

### 5.2.1 Detailed Description

class for Gaussian pulses in 2D

They are given in the form  $\vec{E} = A \vec{\epsilon} \sqrt{\frac{\omega_0}{\omega(z)}} \exp(-r/\omega(z))^2 \exp\left(-((z_g - \Phi_0)/\Phi_A)^2\right) \cos\left(\frac{k r^2}{2R(z)} + g(z) - k z_g\right)$  with

- propagation direction (subtracted distance to origin)  $z_g$
- radial distance to propagation axis  $r = \sqrt{\vec{x}^2 - z_g^2}$
- $k = 2\pi/\lambda$
- waist at position  $z$ ,  $\omega(z) = w_0 \sqrt{1 + (z_g/z_R)^2}$
- Gouy phase  $g(z) = \tan^{-1}(z_g/z_r)$
- beam curvature  $R(z) = z_g (1 + (z_r/z_g)^2)$  obtained via the chosen parameters

Definition at line 139 of file [ICSetters.h](#).

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 Gauss2D()

```
Gauss2D::Gauss2D (
    std::array< sunrealtype, 3 > dis_ = {0, 0, 0},
    std::array< sunrealtype, 3 > axis_ = {1, 0, 0},
    sunrealtype Amp_ = 1.0,
    sunrealtype phip_ = 0,
    sunrealtype w0_ = 1e-5,
    sunrealtype zr_ = 4e-5,
    sunrealtype Ph0_ = 2e-5,
    sunrealtype PhA_ = 0.45e-5 )
```

construction with default parameters

[Gauss2D](#) construction with

- center it approaches
- direction from where it comes
- amplitude
- polarization rotation from TE-mode
- taille
- Rayleigh length
- beam center
- beam length

Definition at line 172 of file [ICSetters.cpp](#).

```

00175
00176     dis = dis_;           /** - center it approaches */
00177     axis = axis_;         /** - direction form where it comes */
00178     Amp = Amp_;          /** - amplitude */
00179     phip = phip_;        /** - polarization rotation from TE-mode */
00180     w0 = w0_;            /** - taille */
00181     zr = zr_;            /** - Rayleigh length */
00182     Ph0 = Ph0_;          /** - beam center */
00183     PhA = PhA_;          /** - beam length */
00184     A1 = Amp * cos(phip); // amplitude in z-direction
00185     A2 = Amp * sin(phip); // amplitude on xy-plane
00186     lambda = std::numbers::pi * w0 * w0 / zr; // formula for wavelength
00187 }
```

References [A1](#), [A2](#), [Amp](#), [axis](#), [dis](#), [lambda](#), [Ph0](#), [PhA](#), [phip](#), [w0](#), and [zr](#).

### 5.2.3 Member Function Documentation

#### 5.2.3.1 addToSpace()

```

void Gauss2D::addToSpace (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space ) const
```

function for the actual implementation in space

Definition at line 189 of file [ICSetters.cpp](#).

```

00190
00191     //\f$ \vec{x} = \vec{x}_0-\vec{dis} \f$ // coordinates minus distance to
00192     //origin
00193     x -= dis[0];
00194     y -= dis[1];
00195     // z-=dis[2];
00196     z = nan("0x12345"); // unused parameter
00197     // \f$ z_g = \vec{x}\cdot\vec{e}_g \f$ projection on propagation axis
00198     const sunrealtype zg =
00199         x * axis[0] + y * axis[1]; // +z*axis[2]; // =z-z0 -> propagation
00200         //direction, minus origin
00201     // \f$ r = \sqrt{\vec{x}^2 - z_g^2} \f$ -> pythagoras of radius minus
00202     // projection on prop axis
00203     const sunrealtype r = sqrt((x * x + y * y /*+z*z*/)
00204         - zg * zg); // radial distance to propagation axis
00205     // \f$ w(z) = w_0\sqrt{1+(z_g/z_R)^2} \f$-
00206     // waist at position z
00207     const sunrealtype wz = w0 * sqrt(1 + (zg * zg / zr / zr));
00208     // \f$ g(z) = \arctan(z_g/z_r) \f$-
00209     const sunrealtype gz = atan(zg / zr); // Gouy phase
00210     // \f$ R(z) = z_g*(1+(z_r/z_g)^2) \f$-
00211     sunrealtype Rz = nan("0x12345"); // beam curvature
00212     if (abs(gz) > 1e-15)
00213         Rz = zg * (1 + (zr * zr / zg / zg));
00214     else
00215         Rz = 1e308;
00216     // wavenumber \f$ k = 2\pi/\lambda \f$-
00217     const sunrealtype k = 2 * std::numbers::pi / lambda;
00218     // \f$ \Phi_F = kr^2/(2*R(z))+g(z)-kz_g \f$-
00219     const sunrealtype PhF =
00220         -k * r * r / (2 * Rz) + gz - k * zg; // to be inserted into cosine
00221     // \f$ G = \sqrt{w_0/w_z}e^{-(r/w(z))^2}e^{-(zg-\Phi_0)^2/PhA^2}\cos(\Phi_F) \f$-
00222     // CVode is a diva, no chance to remove the square in the second exponential
00223     // -> h too small
00224     const sunrealtype G2D = sqrt(w0 / wz) * exp(-r * r / wz / wz) *
00225         exp(-(zg - Ph0) * (zg - Ph0) / PhA / PhA) *
00226         cos(PhF); // gauss shape
00227     // \f$ c_\alpha = \vec{e}_x\cdot\vec{e}_z \f$-
00228     // projection components; do like this for CVode convergence -> otherwise
00229     // results in machine error values for non-existant field components if
00230     // axis[0] and axis[1] are given
```

```

00231 const sunrealtype ca =
00232     axis[0]; // x-component of propagation axis which is given as parameter
00233 // no z-component for 2D propagation
00234 const sunrealtype sa = sqrt(1 - ca * ca);
00235 // E-field to space: polarization in xy-plane (A2) is projection of
00236 // z-polarization (A1) on x- and y-directions
00237 pTo6Space[0] += sa * (G2D * A2);
00238 pTo6Space[1] += -ca * (G2D * A2);
00239 pTo6Space[2] += G2D * A1;
00240 // B-field -> negative derivative wrt polarization shift of E-field
00241 pTo6Space[3] += -sa * (G2D * A1);
00242 pTo6Space[4] += ca * (G2D * A1);
00243 pTo6Space[5] += G2D * A2;
00244 }

```

References [A1](#), [A2](#), [axis](#), [dis](#), [lambda](#), [Ph0](#), [PhA](#), [w0](#), and [zr](#).

## 5.2.4 Field Documentation

### 5.2.4.1 A1

`sunrealtype Gauss2D::A1 [private]`

amplitude projection on TE-mode

Definition at line [159](#) of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

### 5.2.4.2 A2

`sunrealtype Gauss2D::A2 [private]`

amplitude projection on xy-plane

Definition at line [161](#) of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

### 5.2.4.3 Amp

`sunrealtype Gauss2D::Amp [private]`

amplitude  $A$

Definition at line [146](#) of file [ICSetters.h](#).

Referenced by [Gauss2D\(\)](#).

#### 5.2.4.4 axis

```
std::array<sunrealtype, 3> Gauss2D::axis [private]
```

normalized propagation axis

Definition at line 144 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

#### 5.2.4.5 dis

```
std::array<sunrealtype, 3> Gauss2D::dis [private]
```

distance maximum to origin

Definition at line 142 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

#### 5.2.4.6 lambda

```
sunrealtype Gauss2D::lambda [private]
```

wavelength  $\lambda$

Definition at line 163 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

#### 5.2.4.7 Ph0

```
sunrealtype Gauss2D::Ph0 [private]
```

center of beam  $\Phi_0$

Definition at line 155 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

### 5.2.4.8 PhA

`sunrealtype Gauss2D::PhA [private]`

length of beam  $\Phi_A$

Definition at line 157 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

### 5.2.4.9 phip

`sunrealtype Gauss2D::phip [private]`

polarization rotation from TE-mode around propagation direction

Definition at line 149 of file [ICSetters.h](#).

Referenced by [Gauss2D\(\)](#).

### 5.2.4.10 w0

`sunrealtype Gauss2D::w0 [private]`

taille  $\omega_0$

Definition at line 151 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

### 5.2.4.11 zr

`sunrealtype Gauss2D::zr [private]`

Rayleigh length  $z_R = \pi\omega_0^2/\lambda$ .

Definition at line 153 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss2D\(\)](#).

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

- [src/ICSetters.h](#)
- [src/ICSetters.cpp](#)

## 5.3 Gauss3D Class Reference

class for Gaussian pulses in 3D

```
#include <src/ICSetters.h>
```

### Public Member Functions

- `Gauss3D (std::array< sunrealtype, 3 > dis_{0, 0, 0}, std::array< sunrealtype, 3 > axis_{1, 0, 0}, sunrealtype Amp_=1.0, sunrealtype phip_=0, sunrealtype w0_=1e-5, sunrealtype zr_=4e-5, sunrealtype Ph0_=2e-5, sunrealtype PhA_=0.45e-5)`  
*construction with default parameters*
- void `addToSpace (sunrealtype x, sunrealtype y, sunrealtype z, sunrealtype *pTo6Space) const`  
*function for the actual implementation in space*

### Private Attributes

- `std::array< sunrealtype, 3 > dis`  
*distance maximum to origin*
- `std::array< sunrealtype, 3 > axis`  
*normalized propagation axis*
- `sunrealtype Amp`  
*amplitude A*
- `sunrealtype phip`  
*polarization rotation from TE-mode around propagation direction*
- `sunrealtype w0`  
*taille  $\omega_0$*
- `sunrealtype zr`  
*Rayleigh length  $z_R = \pi\omega_0^2/\lambda$ .*
- `sunrealtype Ph0`  
*center of beam  $\Phi_0$*
- `sunrealtype PhA`  
*length of beam  $\Phi_A$*
- `sunrealtype A1`  
*amplitude projection on TE-mode (z-axis)*
- `sunrealtype A2`  
*amplitude projection on xy-plane*
- `sunrealtype lambda`  
*wavelength  $\lambda$*

### 5.3.1 Detailed Description

class for Gaussian pulses in 3D

They are given in the form  $\vec{E} = A \vec{\epsilon} \frac{\omega_0}{\omega(z)} \exp(-r/\omega(z))^2 \exp\left(-((z_g - \Phi_0)/\Phi_A)^2\right) \cos\left(\frac{k r^2}{2R(z)} + g(z) - k z_g\right)$  with

- propagation direction (subtracted distance to origin)  $z_g$
- radial distance to propagation axis  $r = \sqrt{\vec{x}^2 - z_g^2}$
- $k = 2\pi/\lambda$
- waist at position  $z$ ,  $\omega(z) = w_0 \sqrt{1 + (z_g/z_R)^2}$
- Gouy phase  $g(z) = \tan^{-1}(z_g/z_r)$
- beam curvature  $R(z) = z_g (1 + (z_r/z_g)^2)$  obtained via the chosen parameters

Definition at line 193 of file [ICSetters.h](#).

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 Gauss3D()

```
Gauss3D::Gauss3D (
    std::array< sunrealtype, 3 > dis_ = {0, 0, 0},
    std::array< sunrealtype, 3 > axis_ = {1, 0, 0},
    sunrealtype Amp_ = 1.0,
    sunrealtype phip_ = 0,
    sunrealtype w0_ = 1e-5,
    sunrealtype zr_ = 4e-5,
    sunrealtype Ph0_ = 2e-5,
    sunrealtype PhA_ = 0.45e-5 )
```

construction with default parameters

[Gauss3D](#) construction with

- center it approaches
- direction from where it comes
- amplitude
- polarization rotation form TE-mode
- taille
- Rayleigh length
- beam center
- beam length

Definition at line 247 of file [ICSetters.cpp](#).

```

00252                                     {
00253     dis = dis_;    /** - center it approaches */
00254     axis = axis_; /** - direction from where it comes */
00255     Amp = Amp_;   /** - amplitude */
00256     // pol=pol_;
00257     phip = phip_; /** - polarization rotation form TE-mode */
00258     w0 = w0_;      /** - taille */
00259     zr = zr_;       /** - Rayleigh length */
00260     Ph0 = Ph0_;    /** - beam center */
00261     PhA = PhA_;   /** - beam length */
00262     lambda = std::numbers::pi * w0 * w0 / zr;
00263     A1 = Amp * cos(phip);
00264     A2 = Amp * sin(phip);
00265 }
```

References [A1](#), [A2](#), [Amp](#), [axis](#), [dis](#), [lambda](#), [Ph0](#), [PhA](#), [phip](#), [w0](#), and [zr](#).

### 5.3.3 Member Function Documentation

#### 5.3.3.1 addToSpace()

```

void Gauss3D::addToSpace (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space ) const
```

function for the actual implementation in space

[Gauss3D](#) implementation in space

Definition at line 268 of file [ICSetters.cpp](#).

```

00269                                     {
00270     x -= dis[0];
00271     y -= dis[1];
00272     z -= dis[2];
00273     const sunrealtype zg = x * axis[0] + y * axis[1] + z * axis[2];
00274     const sunrealtype r = sqrt((x * x + y * y + z * z) - zg * zg);
00275     const sunrealtype wz = w0 * sqrt(1 + (zg * zg / zr / zr));
00276     const sunrealtype gz = atan(zg / zr);
00277     sunrealtype Rz = nan("0x12345");
00278     if (abs(zg) > 1e-15)
00279         Rz = zg * (1 + (zr * zr / zg / zg));
00280     else
00281         Rz = 1e308;
00282     const sunrealtype k = 2 * std::numbers::pi / lambda;
00283     const sunrealtype PhF = -k * r * r / (2 * Rz) + gz - k * zg;
00284     const sunrealtype G3D = (w0 / wz) * exp(-r * r / wz / wz) *
00285         exp(-(zg - Ph0) * (zg - Ph0) / PhA / PhA) * cos(PhF);
00286     const sunrealtype ca = axis[0];
00287     const sunrealtype sa = sqrt(1 - ca * ca);
00288     pTo6Space[0] += sa * (G3D * A2);
00289     pTo6Space[1] += -ca * (G3D * A2);
00290     pTo6Space[2] += G3D * A1;
00291     pTo6Space[3] += -sa * (G3D * A1);
00292     pTo6Space[4] += ca * (G3D * A1);
00293     pTo6Space[5] += G3D * A2;
00294 }
```

References [A1](#), [A2](#), [axis](#), [dis](#), [lambda](#), [Ph0](#), [PhA](#), [w0](#), and [zr](#).

### 5.3.4 Field Documentation

### 5.3.4.1 A1

`sunrealtype Gauss3D::A1 [private]`

amplitude projection on TE-mode (z-axis)

Definition at line 215 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

### 5.3.4.2 A2

`sunrealtype Gauss3D::A2 [private]`

amplitude projection on xy-plane

Definition at line 217 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

### 5.3.4.3 Amp

`sunrealtype Gauss3D::Amp [private]`

amplitude  $A$

Definition at line 200 of file [ICSetters.h](#).

Referenced by [Gauss3D\(\)](#).

### 5.3.4.4 axis

`std::array<sunrealtype, 3> Gauss3D::axis [private]`

normalized propagation axis

Definition at line 198 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

#### 5.3.4.5 dis

std::array<sunrealtype, 3> Gauss3D::dis [private]

distance maximum to origin

Definition at line 196 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

#### 5.3.4.6 lambda

sunrealtype Gauss3D::lambda [private]

wavelength  $\lambda$

Definition at line 219 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

#### 5.3.4.7 Ph0

sunrealtype Gauss3D::Ph0 [private]

center of beam  $\Phi_0$

Definition at line 211 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

#### 5.3.4.8 PhA

sunrealtype Gauss3D::PhA [private]

length of beam  $\Phi_A$

Definition at line 213 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

### 5.3.4.9 `phip`

`sunrealtype Gauss3D::phip [private]`

polarization rotation from TE-mode around propagation direction

Definition at line 203 of file [ICSetters.h](#).

Referenced by [Gauss3D\(\)](#).

### 5.3.4.10 `w0`

`sunrealtype Gauss3D::w0 [private]`

taille  $\omega_0$

Definition at line 207 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

### 5.3.4.11 `zr`

`sunrealtype Gauss3D::zr [private]`

Rayleigh length  $z_R = \pi\omega_0^2/\lambda$ .

Definition at line 209 of file [ICSetters.h](#).

Referenced by [addToSpace\(\)](#), and [Gauss3D\(\)](#).

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

- `src/ICSetters.h`
- `src/ICSetters.cpp`

## 5.4 gaussian1D Struct Reference

1D Gaussian wave structure

```
#include <src/SimulationFunctions.h>
```

### Data Fields

- `std::array<sunrealtype, 3> k`
- `std::array<sunrealtype, 3> p`
- `std::array<sunrealtype, 3> x0`
- `sunrealtype phig`
- `std::array<sunrealtype, 3> phi`

### 5.4.1 Detailed Description

1D Gaussian wave structure

Definition at line 26 of file [SimulationFunctions.h](#).

### 5.4.2 Field Documentation

#### 5.4.2.1 k

`std::array<sunrealtype, 3> gaussian1D::k`

wavevector (normalized to  $1/\lambda$ )

Definition at line 27 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

#### 5.4.2.2 p

`std::array<sunrealtype, 3> gaussian1D::p`

amplitude & polarization vector

Definition at line 28 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

#### 5.4.2.3 phi

`std::array<sunrealtype, 3> gaussian1D::phi`

phase shift

Definition at line 31 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

#### 5.4.2.4 phig

sunrealtype gaussian1D::phig

width

Definition at line 30 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

#### 5.4.2.5 x0

std::array<sunrealtype, 3> gaussian1D::x0

shift from origin

Definition at line 29 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

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

- src/[SimulationFunctions.h](#)

## 5.5 gaussian2D Struct Reference

2D Gaussian wave structure

```
#include <src/SimulationFunctions.h>
```

### Data Fields

- std::array<sunrealtype, 3> x0
- std::array<sunrealtype, 3> axis
- unrealtype amp
- unrealtype phip
- unrealtype w0
- unrealtype zr
- unrealtype ph0
- unrealtype phA

### 5.5.1 Detailed Description

2D Gaussian wave structure

Definition at line 35 of file [SimulationFunctions.h](#).

## 5.5.2 Field Documentation

### 5.5.2.1 amp

`sunrealtype gaussian2D::amp`

amplitude

Definition at line 38 of file [SimulationFunctions.h](#).

### 5.5.2.2 axis

`std::array<sunrealtype, 3> gaussian2D::axis`

direction from where it comes

Definition at line 37 of file [SimulationFunctions.h](#).

### 5.5.2.3 ph0

`sunrealtype gaussian2D::ph0`

beam center

Definition at line 42 of file [SimulationFunctions.h](#).

### 5.5.2.4 phA

`sunrealtype gaussian2D::phA`

beam length

Definition at line 43 of file [SimulationFunctions.h](#).

### 5.5.2.5 phip

`sunrealtype gaussian2D::phip`

polarization rotation

Definition at line 39 of file [SimulationFunctions.h](#).

### 5.5.2.6 w0

sunrealtype gaussian2D::w0

taille

Definition at line 40 of file [SimulationFunctions.h](#).

### 5.5.2.7 x0

std::array<sunrealtype, 3> gaussian2D::x0

center

Definition at line 36 of file [SimulationFunctions.h](#).

### 5.5.2.8 zr

sunrealtype gaussian2D::zr

Rayleigh length

Definition at line 41 of file [SimulationFunctions.h](#).

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

- src/[SimulationFunctions.h](#)

## 5.6 gaussian3D Struct Reference

3D Gaussian wave structure

```
#include <src/SimulationFunctions.h>
```

### Data Fields

- std::array<sunrealtype, 3> x0
- std::array<sunrealtype, 3> axis
- unrealtype amp
- unrealtype phip
- unrealtype w0
- unrealtype zr
- unrealtype ph0
- unrealtype phA

### 5.6.1 Detailed Description

3D Gaussian wave structure

Definition at line 47 of file [SimulationFunctions.h](#).

### 5.6.2 Field Documentation

#### 5.6.2.1 amp

```
sunrealtype gaussian3D::amp
```

amplitude

Definition at line 50 of file [SimulationFunctions.h](#).

#### 5.6.2.2 axis

```
std::array<sunrealtype, 3> gaussian3D::axis
```

direction from where it comes

Definition at line 49 of file [SimulationFunctions.h](#).

#### 5.6.2.3 ph0

```
sunrealtype gaussian3D::ph0
```

beam center

Definition at line 54 of file [SimulationFunctions.h](#).

#### 5.6.2.4 phA

```
sunrealtype gaussian3D::phA
```

beam length

Definition at line 55 of file [SimulationFunctions.h](#).

### 5.6.2.5 phip

sunrealtype gaussian3D::phip

polarization rotation

Definition at line 51 of file [SimulationFunctions.h](#).

### 5.6.2.6 w0

sunrealtype gaussian3D::w0

taille

Definition at line 52 of file [SimulationFunctions.h](#).

### 5.6.2.7 x0

std::array<sunrealtype, 3> gaussian3D::x0

center

Definition at line 48 of file [SimulationFunctions.h](#).

### 5.6.2.8 zr

sunrealtype gaussian3D::zr

Rayleigh length

Definition at line 53 of file [SimulationFunctions.h](#).

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

- src/[SimulationFunctions.h](#)

## 5.7 ICSetter Class Reference

[ICSetter](#) class to initialize wave types with default parameters.

```
#include <src/ICSetters.h>
```

## Public Member Functions

- void [eval](#) (sunrealtype x, unrealtype y, unrealtype z, unrealtype \*pTo6Space)  
*function to set all coordinates to zero and then add the field values*
- void [add](#) (unrealtype x, unrealtype y, unrealtype z, unrealtype \*pTo6Space)  
*function to fill the lattice space with initial field values*
- void [addPlaneWave1D](#) (std::array< unrealtype, 3 > k={1, 0, 0}, std::array< unrealtype, 3 > p={0, 0, 1}, std::array< unrealtype, 3 > phi={0, 0, 0})  
*function to add plane waves in 1D to their container vector*
- void [addPlaneWave2D](#) (std::array< unrealtype, 3 > k={1, 0, 0}, std::array< unrealtype, 3 > p={0, 0, 1}, std::array< unrealtype, 3 > phi={0, 0, 0})  
*function to add plane waves in 2D to their container vector*
- void [addPlaneWave3D](#) (std::array< unrealtype, 3 > k={1, 0, 0}, std::array< unrealtype, 3 > p={0, 0, 1}, std::array< unrealtype, 3 > phi={0, 0, 0})  
*function to add plane waves in 3D to their container vector*
- void [addGauss1D](#) (std::array< unrealtype, 3 > k={1, 0, 0}, std::array< unrealtype, 3 > p={0, 0, 1}, std::array< unrealtype, 3 > xo={0, 0, 0}, unrealtype phig\_=1.0, std::array< unrealtype, 3 > phi={0, 0, 0})  
*function to add Gaussian wave packets in 1D to their container vector*
- void [addGauss2D](#) (std::array< unrealtype, 3 > dis\_={0, 0, 0}, std::array< unrealtype, 3 > axis\_={1, 0, 0}, unrealtype Amp\_=1.0, unrealtype phip\_=0, unrealtype w0\_=1e-5, unrealtype zr\_=4e-5, unrealtype Ph0\_=2e-5, unrealtype PhA\_=0.45e-5)  
*function to add Gaussian wave packets in 2D to their container vector*
- void [addGauss3D](#) (std::array< unrealtype, 3 > dis\_={0, 0, 0}, std::array< unrealtype, 3 > axis\_={1, 0, 0}, unrealtype Amp\_=1.0, unrealtype phip\_=0, unrealtype w0\_=1e-5, unrealtype zr\_=4e-5, unrealtype Ph0\_=2e-5, unrealtype PhA\_=0.45e-5)  
*function to add Gaussian wave packets in 3D to their container vector*

## Private Attributes

- std::vector< [PlaneWave1D](#) > [planeWaves1D](#)  
*container vector for plane waves in 1D*
- std::vector< [PlaneWave2D](#) > [planeWaves2D](#)  
*container vector for plane waves in 2D*
- std::vector< [PlaneWave3D](#) > [planeWaves3D](#)  
*container vector for plane waves in 3D*
- std::vector< [Gauss1D](#) > [gauss1Ds](#)  
*container vector for Gaussian wave packets in 1D*
- std::vector< [Gauss2D](#) > [gauss2Ds](#)  
*container vector for Gaussian wave packets in 2D*
- std::vector< [Gauss3D](#) > [gauss3Ds](#)  
*container vector for Gaussian wave packets in 3D*

### 5.7.1 Detailed Description

[ICSetter](#) class to initialize wave types with default parameters.

Definition at line 238 of file [ICSetters.h](#).

### 5.7.2 Member Function Documentation

### 5.7.2.1 add()

```
void ICSetter::add (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space )
```

function to fill the lattice space with initial field values

Add all initial field values to the lattice space

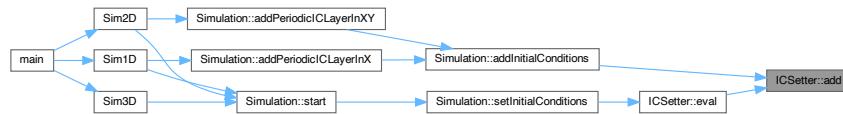
Definition at line 309 of file [ICSetters.cpp](#).

```
00310
00311     for (const auto &wave : planeWaves1D)
00312         wave.addToSpace(x, y, z, pTo6Space);
00313     for (const auto &wave : planeWaves2D)
00314         wave.addToSpace(x, y, z, pTo6Space);
00315     for (const auto &wave : planeWaves3D)
00316         wave.addToSpace(x, y, z, pTo6Space);
00317     for (const auto &wave : gauss1Ds)
00318         wave.addToSpace(x, y, z, pTo6Space);
00319     for (const auto &wave : gauss2Ds)
00320         wave.addToSpace(x, y, z, pTo6Space);
00321     for (const auto &wave : gauss3Ds)
00322         wave.addToSpace(x, y, z, pTo6Space);
00323 }
```

References [gauss1Ds](#), [gauss2Ds](#), [gauss3Ds](#), [planeWaves1D](#), [planeWaves2D](#), and [planeWaves3D](#).

Referenced by [Simulation::addInitialConditions\(\)](#), and [eval\(\)](#).

Here is the caller graph for this function:



### 5.7.2.2 addGauss1D()

```
void ICSetter::addGauss1D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > xo = {0, 0, 0},
    sunrealtype phig_ = 1.0,
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

function to add Gaussian wave packets in 1D to their container vector

Add Gaussian waves in 1D to their container vector

Definition at line 347 of file [ICSetters.cpp](#).

```
00350
00351     gauss1Ds.emplace_back(Gauss1D(k, p, xo, phig_, phi));
00352 }
```

References [gauss1Ds](#).

Referenced by [Sim1D\(\)](#).

Here is the caller graph for this function:



### 5.7.2.3 addGauss2D()

```
void ICSetter::addGauss2D (
    std::array< sunrealtype, 3 > dis_ = {0, 0, 0},
    std::array< sunrealtype, 3 > axis_ = {1, 0, 0},
    sunrealtype Amp_ = 1.0,
    sunrealtype phip_ = 0,
    sunrealtype w0_ = 1e-5,
    sunrealtype zr_ = 4e-5,
    sunrealtype Ph0_ = 2e-5,
    sunrealtype PhA_ = 0.45e-5 )
```

function to add Gaussian wave packets in 2D to their container vector

Add Gaussian waves in 2D to their container vector

Definition at line 355 of file [ICSetters.cpp](#).

```
00359 {
00360     gauss2Ds.emplace_back(
00361         Gauss2D(dis_, axis_, Amp_, phip_, w0_, zr_, Ph0_, PhA_));
00362 }
```

References [gauss2Ds](#).

Referenced by [Sim2D\(\)](#).

Here is the caller graph for this function:



### 5.7.2.4 addGauss3D()

```
void ICSetter::addGauss3D (
    std::array< sunrealtype, 3 > dis_ = {0, 0, 0},
    std::array< sunrealtype, 3 > axis_ = {1, 0, 0},
    sunrealtype Amp_ = 1.0,
    sunrealtype phip_ = 0,
    sunrealtype w0_ = 1e-5,
    sunrealtype zr_ = 4e-5,
    sunrealtype Ph0_ = 2e-5,
    sunrealtype PhA_ = 0.45e-5 )
```

function to add Gaussian wave packets in 3D to their container vector

Add Gaussian waves in 3D to their container vector

Definition at line 365 of file [ICSetters.cpp](#).

```
00369 {
00370     gauss3Ds.emplace_back(
00371         Gauss3D(dis_, axis_, Amp_, phip_, w0_, zr_, Ph0_, PhA_));
00372 }
```

References [gauss3Ds](#).

Referenced by [Sim3D\(\)](#).

Here is the caller graph for this function:



### 5.7.2.5 addPlaneWave1D()

```
void ICSetter::addPlaneWave1D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

function to add plane waves in 1D to their container vector

Add plane waves in 1D to their container vector

Definition at line 326 of file [ICSetters.cpp](#).

```
00328 {
00329     planeWaves1D.emplace_back(PlaneWave1D(k, p, phi));
00330 }
```

References [planeWaves1D](#).

Referenced by [Sim1D\(\)](#).

Here is the caller graph for this function:



### 5.7.2.6 addPlaneWave2D()

```
void ICSetter::addPlaneWave2D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

function to add plane waves in 2D to their container vector

Add plane waves in 2D to their container vector

Definition at line 333 of file [ICSetters.cpp](#).

```
00335 {
00336     planeWaves2D.emplace_back(PlaneWave2D(k, p, phi));
00337 }
```

References [planeWaves2D](#).

Referenced by [Sim2D\(\)](#).

Here is the caller graph for this function:



### 5.7.2.7 addPlaneWave3D()

```
void ICSetter::addPlaneWave3D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

function to add plane waves in 3D to their container vector

Add plane waves in 3D to their container vector

Definition at line 340 of file [ICSetters.cpp](#).

```
00342                                     {
00343     planeWaves3D.emplace_back(PlaneWave3D(k, p, phi));
00344 }
```

References [planeWaves3D](#).

Referenced by [Sim3D\(\)](#).

Here is the caller graph for this function:



### 5.7.2.8 eval()

```
void ICSetter::eval (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space )
```

function to set all coordinates to zero and then add the field values

Evaluate lattice point values to zero and then add initial field values

Definition at line 297 of file [ICSetters.cpp](#).

```
00298                                     {
00299     pTo6Space[0] = 0;
00300     pTo6Space[1] = 0;
00301     pTo6Space[2] = 0;
00302     pTo6Space[3] = 0;
00303     pTo6Space[4] = 0;
00304     pTo6Space[5] = 0;
00305     add(x, y, z, pTo6Space);
00306 }
```

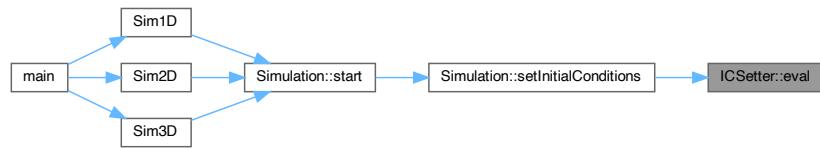
References [add\(\)](#).

Referenced by [Simulation::setInitialConditions\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.7.3 Field Documentation

#### 5.7.3.1 gauss1Ds

```
std::vector<Gauss1D> ICSetter::gauss1Ds [private]
```

container vector for Gaussian wave packets in 1D

Definition at line [247](#) of file [ICSetters.h](#).

Referenced by [add\(\)](#), and [addGauss1D\(\)](#).

#### 5.7.3.2 gauss2Ds

```
std::vector<Gauss2D> ICSetter::gauss2Ds [private]
```

container vector for Gaussian wave packets in 2D

Definition at line [249](#) of file [ICSetters.h](#).

Referenced by [add\(\)](#), and [addGauss2D\(\)](#).

### 5.7.3.3 gauss3Ds

```
std::vector<Gauss3D> ICSetter::gauss3Ds [private]
```

container vector for Gaussian wave packets in 3D

Definition at line [251](#) of file [ICSetters.h](#).

Referenced by [add\(\)](#), and [addGauss3D\(\)](#).

### 5.7.3.4 planeWaves1D

```
std::vector<PlaneWave1D> ICSetter::planeWaves1D [private]
```

container vector for plane waves in 1D

Definition at line [241](#) of file [ICSetters.h](#).

Referenced by [add\(\)](#), and [addPlaneWave1D\(\)](#).

### 5.7.3.5 planeWaves2D

```
std::vector<PlaneWave2D> ICSetter::planeWaves2D [private]
```

container vector for plane waves in 2D

Definition at line [243](#) of file [ICSetters.h](#).

Referenced by [add\(\)](#), and [addPlaneWave2D\(\)](#).

### 5.7.3.6 planeWaves3D

```
std::vector<PlaneWave3D> ICSetter::planeWaves3D [private]
```

container vector for plane waves in 3D

Definition at line [245](#) of file [ICSetters.h](#).

Referenced by [add\(\)](#), and [addPlaneWave3D\(\)](#).

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

- [src/ICSetters.h](#)
- [src/ICSetters.cpp](#)

## 5.8 Lattice Class Reference

[Lattice](#) class for the construction of the enveloping discrete simulation space.

```
#include <src/LatticePatch.h>
```

### Public Member Functions

- void [initializeCommunicator](#) (const int Nx, const int Ny, const int Nz, const bool per)  
*function to create and deploy the cartesian communicator*
  - [Lattice](#) (const int StO)  
*default construction*
  - void [setDiscreteDimensions](#) (const sunindextype \_nx, const sunindextype \_ny, const sunindextype \_nz)  
*component function for resizing the discrete dimensions of the lattice*
  - void [setPhysicalDimensions](#) (const sunrealtype \_lx, const sunrealtype \_ly, const sunrealtype \_lz)  
*component function for resizing the physical size of the lattice*
- 
- const sunrealtype & [get\\_tot\\_lx](#) () const
  - const sunrealtype & [get\\_tot\\_ly](#) () const
  - const sunrealtype & [get\\_tot\\_lz](#) () const
  - const sunindextype & [get\\_tot\\_nx](#) () const
  - const sunindextype & [get\\_tot\\_ny](#) () const
  - const sunindextype & [get\\_tot\\_nz](#) () const
  - const sunindextype & [get\\_tot\\_noP](#) () const
  - const sunindextype & [get\\_tot\\_noDP](#) () const
  - const sunrealtype & [get\\_dx](#) () const
  - const sunrealtype & [get\\_dy](#) () const
  - const sunrealtype & [get\\_dz](#) () const
  - constexpr int [get\\_dataPointDimension](#) () const
  - const int & [get\\_stencilOrder](#) () const
  - const int & [get\\_ghostLayerWidth](#) () const

### Data Fields

- int [n\\_prc](#)  
*number of MPI processes*
- int [my\\_prc](#)  
*number of MPI process*
- MPI\_Comm [comm](#)  
*personal communicator of the lattice*
- SUNContext [suncxt](#)  
*SUNContext object.*

## Private Attributes

- sunrealtype `tot_lx`  
*physical size of the lattice in x-direction*
- sunrealtype `tot_ly`  
*physical size of the lattice in y-direction*
- sunrealtype `tot_lz`  
*physical size of the lattice in z-direction*
- sunindextype `tot_nx`  
*number of points in x-direction*
- sunindextype `tot_ny`  
*number of points in y-direction*
- sunindextype `tot_nz`  
*number of points in z-direction*
- sunindextype `tot_noP`  
*total number of lattice points*
- sunindextype `tot_noDP`  
*number of lattice points times data dimension of each point*
- sunrealtype `dx`  
*physical distance between lattice points in x-direction*
- sunrealtype `dy`  
*physical distance between lattice points in y-direction*
- sunrealtype `dz`  
*physical distance between lattice points in z-direction*
- const int `StencilOrder`  
*stencil order*
- const int `ghostLayerWidth`  
*required width of ghost layers (depends on the stencil order)*
- unsigned int `statusFlags`  
*lattice status flags*

## Static Private Attributes

- static constexpr int `dataPointDimension` = 6  
*dimension of each data point set once and for all*

### 5.8.1 Detailed Description

[Lattice](#) class for the construction of the enveloping discrete simulation space.

Definition at line 52 of file [LatticePatch.h](#).

### 5.8.2 Constructor & Destructor Documentation

### 5.8.2.1 Lattice()

```
Lattice::Lattice (
    const int Sto )
```

default construction

Construct the lattice and set the stencil order.

Definition at line 32 of file [LatticePatch.cpp](#).

```
00032     : stencilOrder(Sto),
00033     ghostLayerWidth(Sto/2+1) {
00034     statusFlags = 0;
00035 }
```

References [statusFlags](#).

## 5.8.3 Member Function Documentation

### 5.8.3.1 get\_dataPointDimension()

```
constexpr int Lattice::get_dataPointDimension ( ) const [inline], [constexpr]
```

getter function

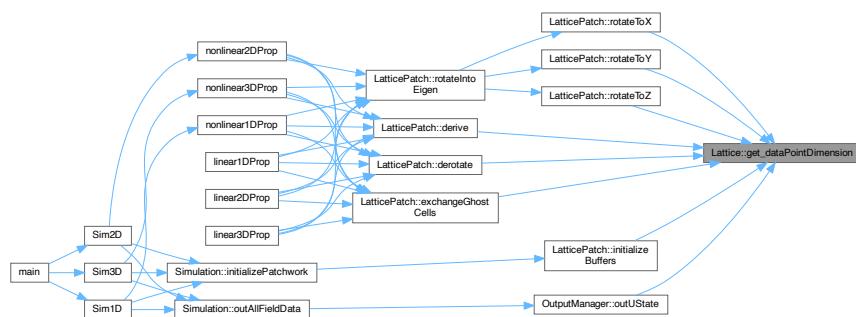
Definition at line 118 of file [LatticePatch.h](#).

```
00118
00119     return dataPointDimension;
00120 }
```

References [dataPointDimension](#).

Referenced by [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), [LatticePatch::initializeBuffers\(\)](#), [OutputManager::outUState\(\)](#), [LatticePatch::rotateToX\(\)](#), [LatticePatch::rotateToY\(\)](#), and [LatticePatch::rotateToZ\(\)](#).

Here is the caller graph for this function:



### 5.8.3.2 get\_dx()

```
const sunrealtype & Lattice::get_dx ( ) const [inline]
```

getter function

Definition at line 115 of file [LatticePatch.h](#).

```
00115 { return dx; }
```

References [dx](#).

### 5.8.3.3 get\_dy()

```
const sunrealtype & Lattice::get_dy ( ) const [inline]
```

getter function

Definition at line 116 of file [LatticePatch.h](#).

```
00116 { return dy; }
```

References [dy](#).

### 5.8.3.4 get\_dz()

```
const sunrealtype & Lattice::get_dz ( ) const [inline]
```

getter function

Definition at line 117 of file [LatticePatch.h](#).

```
00117 { return dz; }
```

References [dz](#).

### 5.8.3.5 get\_ghostLayerWidth()

```
const int & Lattice::get_ghostLayerWidth ( ) const [inline]
```

getter function

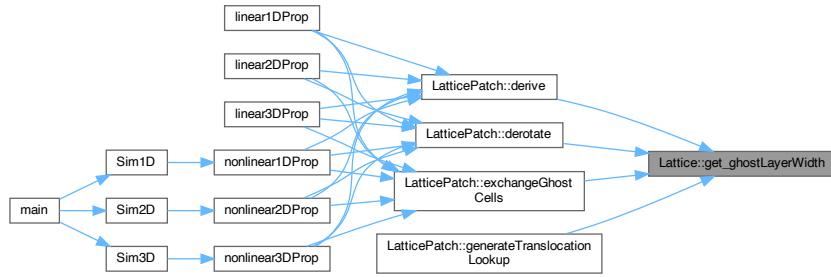
Definition at line 122 of file [LatticePatch.h](#).

```
00122     {  
00123         return ghostLayerWidth;  
00124     }
```

References [ghostLayerWidth](#).

Referenced by [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::generateTranslocationLookup\(\)](#).

Here is the caller graph for this function:



### 5.8.3.6 get\_stencilOrder()

```
const int & Lattice::get_stencilOrder ( ) const [inline]
```

getter function

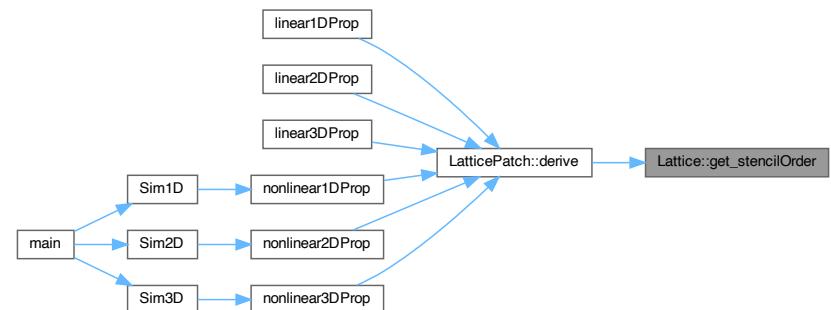
Definition at line 121 of file [LatticePatch.h](#).

```
00121 { return stencilOrder; }
```

References [stencilOrder](#).

Referenced by [LatticePatch::derive\(\)](#).

Here is the caller graph for this function:



### 5.8.3.7 get\_tot\_lx()

```
const sunrealtype & Lattice::get_tot_lx ( ) const [inline]
```

getter function

Definition at line 107 of file [LatticePatch.h](#).

```
00107 { return tot_lx; }
```

References [tot\\_lx](#).

Referenced by [Simulation::addInitialConditions\(\)](#).

Here is the caller graph for this function:



### 5.8.3.8 get\_tot\_ly()

```
const sunrealtype & Lattice::get_tot_ly ( ) const [inline]
```

getter function

Definition at line 108 of file [LatticePatch.h](#).

```
00108 { return tot_ly; }
```

References [tot\\_ly](#).

Referenced by [Simulation::addInitialConditions\(\)](#).

Here is the caller graph for this function:



### 5.8.3.9 get\_tot\_lz()

```
const sunrealtype & Lattice::get_tot_lz ( ) const [inline]
```

getter function

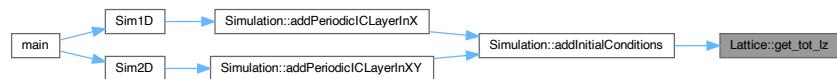
Definition at line 109 of file [LatticePatch.h](#).

```
00109 { return tot_lz; }
```

References [tot\\_lz](#).

Referenced by [Simulation::addInitialConditions\(\)](#).

Here is the caller graph for this function:



### 5.8.3.10 get\_tot\_noDP()

```
const sunindextype & Lattice::get_tot_noDP ( ) const [inline]
```

getter function

Definition at line 114 of file [LatticePatch.h](#).

```
00114 { return tot_noDP; }
```

References [tot\\_noDP](#).

### 5.8.3.11 get\_tot\_noP()

```
const sunindextype & Lattice::get_tot_noP ( ) const [inline]
```

getter function

Definition at line 113 of file [LatticePatch.h](#).

```
00113 { return tot_noP; }
```

References [tot\\_noP](#).

### 5.8.3.12 get\_tot\_nx()

```
const sunindextype & Lattice::get_tot_nx ( ) const [inline]
```

getter function

Definition at line 110 of file [LatticePatch.h](#).  
00110 { **return tot\_nx;** }

References [tot\\_nx](#).

### 5.8.3.13 get\_tot\_ny()

```
const sunindextype & Lattice::get_tot_ny ( ) const [inline]
```

getter function

Definition at line 111 of file [LatticePatch.h](#).  
00111 { **return tot\_ny;** }

References [tot\\_ny](#).

### 5.8.3.14 get\_tot\_nz()

```
const sunindextype & Lattice::get_tot_nz ( ) const [inline]
```

getter function

Definition at line 112 of file [LatticePatch.h](#).  
00112 { **return tot\_nz;** }

References [tot\\_nz](#).

### 5.8.3.15 initializeCommunicator()

```
void Lattice::initializeCommunicator (
    const int Nx,
    const int Ny,
    const int Nz,
    const bool per )
```

function to create and deploy the cartesian communicator

Initialize the cartesian communicator.

Definition at line 15 of file [LatticePatch.cpp](#).

```
00016     {
00017     const int dims[3] = {Nz, Ny, Nx};
00018     const int periods[3] = {static_cast<int>(per), static_cast<int>(per),
00019                             static_cast<int>(per)};
00020     // Create the cartesian communicator for MPI_COMM_WORLD
00021     MPI_Cart_create(MPI_COMM_WORLD, 3, dims, periods, 1, &comm);
00022     // Set MPI variables of the lattice
00023     MPI_Comm_size(comm, &(n_prc));
00024     MPI_Comm_rank(comm, &(my_prc));
00025     // Associate name to the communicator to identify it -> for debugging and
00026     // nicer error messages
00027     constexpr char lattice_comm_name[] = "Lattice";
00028     MPI_Comm_set_name(comm, lattice_comm_name);
00029 }
```

References [comm](#), [my\\_prc](#), and [n\\_prc](#).

Referenced by [Simulation::Simulation\(\)](#).

Here is the caller graph for this function:



### 5.8.3.16 setDiscreteDimensions()

```
void Lattice::setDiscreteDimensions (
    const sunindextype _nx,
    const sunindextype _ny,
    const sunindextype _nz )
```

component function for resizing the discrete dimensions of the lattice

Set the number of points in each dimension of the lattice.

Definition at line 38 of file [LatticePatch.cpp](#).

```
00039     {
00040     // copy the given data for number of points
00041     tot_nx = _nx;
00042     tot_ny = _ny;
```

```

00043     tot_nz = _nz;
00044     // compute the resulting number of points and datapoints
00045     tot_noP = tot_nx * tot_ny * tot_nz;
00046     tot_noDP = dataPointDimension * tot_noP;
00047     // compute the new Delta, the physical resolution
00048     dx = tot_lx / tot_nx;
00049     dy = tot_ly / tot_ny;
00050     dz = tot_lz / tot_nz;
00051 }

```

References [dataPointDimension](#), [dx](#), [dy](#), [dz](#), [tot\\_lx](#), [tot\\_ly](#), [tot\\_lz](#), [tot\\_noDP](#), [tot\\_noP](#), [tot\\_nx](#), [tot\\_ny](#), and [tot\\_nz](#).

Referenced by [Simulation::setDiscreteDimensionsOfLattice\(\)](#).

Here is the caller graph for this function:



### 5.8.3.17 setPhysicalDimensions()

```

void Lattice::setPhysicalDimensions (
    const sunrealtypetot_lx,
    const sunrealtypetot_ly,
    const sunrealtypetot_lz )

```

component function for resizing the physical size of the lattice

Set the physical size of the lattice.

Definition at line 54 of file [LatticePatch.cpp](#).

```

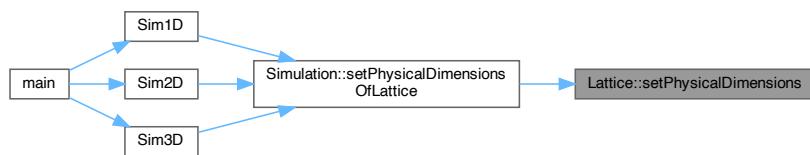
00055     tot_lx = _lx;
00056     tot_ly = _ly;
00057     tot_lz = _lz;
00058     // calculate physical distance between points
00059     dx = tot_lx / tot_nx;
00060     dy = tot_ly / tot_ny;
00061     dz = tot_lz / tot_nz;
00062     statusFlags |= FLatticeDimensionSet;
00063
00064 }

```

References [dx](#), [dy](#), [dz](#), [FLatticeDimensionSet](#), [statusFlags](#), [tot\\_lx](#), [tot\\_ly](#), [tot\\_lz](#), [tot\\_nx](#), [tot\\_ny](#), and [tot\\_nz](#).

Referenced by [Simulation::setPhysicalDimensionsOfLattice\(\)](#).

Here is the caller graph for this function:



## 5.8.4 Field Documentation

### 5.8.4.1 comm

```
MPI_Comm Lattice::comm
```

personal communicator of the lattice

Definition at line 91 of file [LatticePatch.h](#).

Referenced by [Simulation::advanceToTime\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), [Simulation::get\\_cart\\_comm\(\)](#), [initializeCommunicator\(\)](#), [Simulation::initializeCVODEobject\(\)](#), [OutputManager::outUState\(\)](#), and [Simulation::Simulation\(\)](#).

### 5.8.4.2 dataPointDimension

```
constexpr int Lattice::dataPointDimension = 6 [static], [constexpr], [private]
```

dimension of each data point set once and for all

Definition at line 69 of file [LatticePatch.h](#).

Referenced by [get\\_dataPointDimension\(\)](#), and [setDiscreteDimensions\(\)](#).

### 5.8.4.3 dx

```
sunrealtype Lattice::dx [private]
```

physical distance between lattice points in x-direction

Definition at line 73 of file [LatticePatch.h](#).

Referenced by [get\\_dx\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

### 5.8.4.4 dy

```
sunrealtype Lattice::dy [private]
```

physical distance between lattice points in y-direction

Definition at line 75 of file [LatticePatch.h](#).

Referenced by [get\\_dy\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

### 5.8.4.5 dz

```
sunrealtype Lattice::dz [private]
```

physical distance between lattice points in z-direction

Definition at line 77 of file [LatticePatch.h](#).

Referenced by [get\\_dz\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

### 5.8.4.6 ghostLayerWidth

```
const int Lattice::ghostLayerWidth [private]
```

required width of ghost layers (depends on the stencil order)

Definition at line 81 of file [LatticePatch.h](#).

Referenced by [get\\_ghostLayerWidth\(\)](#).

### 5.8.4.7 my\_prc

```
int Lattice::my_prc
```

number of MPI process

Definition at line 89 of file [LatticePatch.h](#).

Referenced by [Simulation::advanceToTime\(\)](#), [initializeCommunicator\(\)](#), [Simulation::initializeCVODEobject\(\)](#), [OutputManager::outUState\(\)](#), and [Simulation::Simulation\(\)](#).

### 5.8.4.8 n\_prc

```
int Lattice::n_prc
```

number of MPI processes

Definition at line 87 of file [LatticePatch.h](#).

Referenced by [initializeCommunicator\(\)](#).

#### 5.8.4.9 statusFlags

```
unsigned int Lattice::statusFlags [private]
```

lattice status flags

Definition at line 83 of file [LatticePatch.h](#).

Referenced by [Lattice\(\)](#), and [setPhysicalDimensions\(\)](#).

#### 5.8.4.10 stencilOrder

```
const int Lattice::stencilOrder [private]
```

stencil order

Definition at line 79 of file [LatticePatch.h](#).

Referenced by [get\\_stencilOrder\(\)](#).

#### 5.8.4.11 sunctx

```
SUNContext Lattice::sunctx
```

SUNContext object.

Definition at line 98 of file [LatticePatch.h](#).

Referenced by [Simulation::initializeCVODEobject\(\)](#), [Simulation::Simulation\(\)](#), and [Simulation::~Simulation\(\)](#).

#### 5.8.4.12 tot\_lx

```
sunrealtype Lattice::tot_lx [private]
```

physical size of the lattice in x-direction

Definition at line 55 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_lx\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

#### 5.8.4.13 tot\_ly

```
sunrealtype Lattice::tot_ly [private]
```

physical size of the lattice in y-direction

Definition at line 57 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_ly\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

#### 5.8.4.14 tot\_lz

```
sunrealtype Lattice::tot_lz [private]
```

physical size of the lattice in z-direction

Definition at line 59 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_lz\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

#### 5.8.4.15 tot\_noDP

```
sunindextype Lattice::tot_noDP [private]
```

number of lattice points times data dimension of each point

Definition at line 71 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_noDP\(\)](#), and [setDiscreteDimensions\(\)](#).

#### 5.8.4.16 tot\_noP

```
sunindextype Lattice::tot_noP [private]
```

total number of lattice points

Definition at line 67 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_noP\(\)](#), and [setDiscreteDimensions\(\)](#).

#### 5.8.4.17 tot\_nx

sunindextype Lattice::tot\_nx [private]

number of points in x-direction

Definition at line 61 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_nx\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

#### 5.8.4.18 tot\_ny

sunindextype Lattice::tot\_ny [private]

number of points in y-direction

Definition at line 63 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_ny\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

#### 5.8.4.19 tot\_nz

sunindextype Lattice::tot\_nz [private]

number of points in z-direction

Definition at line 65 of file [LatticePatch.h](#).

Referenced by [get\\_tot\\_nz\(\)](#), [setDiscreteDimensions\(\)](#), and [setPhysicalDimensions\(\)](#).

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

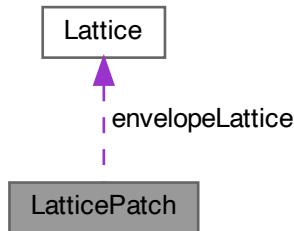
- src/[LatticePatch.h](#)
- src/[LatticePatch.cpp](#)

## 5.9 LatticePatch Class Reference

[LatticePatch](#) class for the construction of the patches in the enveloping lattice.

```
#include <src/LatticePatch.h>
```

Collaboration diagram for LatticePatch:



## Public Member Functions

- **LatticePatch ()**  
*constructor setting up a default first lattice patch*
  - **~LatticePatch ()**  
*destructor freeing parallel vectors*
  - sunindextype **discreteSize** (int dir=0) const  
*function to get the discrete size of the LatticePatch*
  - sunrealtype **origin** (const int dir) const  
*function to get the origin of the patch*
  - sunrealtype **getDelta** (const int dir) const  
*function to get distance between points*
  - void **generateTranslocationLookup ()**  
*function to fill out the lookup tables for cache efficiency*
  - void **rotateIntoEigen** (const int dir)  
*function to rotate u into Z-matrix eigenraum*
  - void **derotate** (int dir, sunrealtype \*buffOut)  
*function to derotate uAux into dudata lattice direction of x*
  - void **initializeBuffers ()**  
*initialize buffers to save derivatives*
  - void **exchangeGhostCells** (const int dir)  
*function to exchange ghost cells*
  - void **derive** (const int dir)  
*function to derive the centered values in uAux and save them noncentered*
  - void **checkFlag** (unsigned int flag) const  
*function to check if a flag has been set and if not abort*

## Data Fields

- int **ID**  
*ID of the [LatticePatch](#), corresponds to process number (for debugging)*
  - N\_Vector **uLocal**  
*NVector for saving field components  $u=(E,B)$  in lattice points.*
  - N\_Vector **u**
  - N\_Vector **duLocal**  
*NVector for saving temporal derivatives of the field data.*
  - N\_Vector **du**
  - sunrealtype \* **uData**  
*pointer to field data*
  - sunrealtype \* **duData**  
*pointer to time-derivative data*
  - sunrealtype \* **uAuxData**  
*pointer to auxiliary data vector*
  - std::array< sunrealtype \*, 3 > **buffData**
  
  - sunrealtype \* **gCLData**
  - sunrealtype \* **gCRData**

## Private Member Functions

- void `rotateToX` (sunrealtype \*outArray, const unrealtype \*inArray, const std::vector< sunindextype > &lookup)
- void `rotateToY` (sunrealtype \*outArray, const unrealtype \*inArray, const std::vector< sunindextype > &lookup)
- void `rotateToZ` (sunrealtype \*outArray, const unrealtype \*inArray, const std::vector< sunindextype > &lookup)

## Private Attributes

- unrealtype `x0`  
*origin of the patch in physical space; x-coordinate*
- unrealtype `y0`  
*origin of the patch in physical space; y-coordinate*
- unrealtype `z0`  
*origin of the patch in physical space; z-coordinate*
- sunindextype `Llx`  
*inner position of lattice-patch in the lattice patchwork; x-points*
- sunindextype `Lly`  
*inner position of lattice-patch in the lattice patchwork; y-points*
- sunindextype `Llz`  
*inner position of lattice-patch in the lattice patchwork; z-points*
- unrealtype `lx`  
*physical size of the lattice-patch in the x-dimension*
- unrealtype `ly`  
*physical size of the lattice-patch in the y-dimension*
- unrealtype `lz`  
*physical size of the lattice-patch in the z-dimension*
- sunindextype `nx`  
*number of points in the lattice patch in the x-dimension*
- sunindextype `ny`  
*number of points in the lattice patch in the y-dimension*
- sunindextype `nz`  
*number of points in the lattice patch in the z-dimension*
- unrealtype `dx`  
*physical distance between lattice points in x-direction*
- unrealtype `dy`  
*physical distance between lattice points in y-direction*
- unrealtype `dz`  
*physical distance between lattice points in z-direction*
- unsigned int `statusFlags`  
*lattice patch status flags*
- const `Lattice` \* `envelopeLattice`  
*pointer to the enveloping lattice*
- std::vector< unrealtype > `uAux`  
*aid (auxilliarly) vector including ghost cells to compute the derivatives*

- std::vector< sunindextype > uTox
  - std::vector< sunindextype > uToy
  - std::vector< sunindextype > uToz
  - std::vector< sunindextype > xTou
  - std::vector< sunindextype > yTou
  - std::vector< sunindextype > zTou
- 
- std::vector< sunrealtype > buffX
  - std::vector< sunrealtype > buffY
  - std::vector< sunrealtype > buffZ
- 
- std::vector< sunrealtype > ghostCellLeft
  - std::vector< sunrealtype > ghostCellRight
  - std::vector< sunrealtype > ghostCellLeftToSend
  - std::vector< sunrealtype > ghostCellRightToSend
  - std::vector< sunrealtype > ghostCellsToSend
  - std::vector< sunrealtype > ghostCells
- 
- std::vector< sunindextype > lgcTox
  - std::vector< sunindextype > rgcTox
  - std::vector< sunindextype > lgcToy
  - std::vector< sunindextype > rgcToy
  - std::vector< sunindextype > lgcToz
  - std::vector< sunindextype > rgcToz

## Friends

- int [generatePatchwork](#) (const [Lattice](#) &envelopeLattice, [LatticePatch](#) &patchToMold, const int DLx, const int DLy, const int DLz)  
*friend function for creating the patchwork slicing of the overall lattice*

### 5.9.1 Detailed Description

[LatticePatch](#) class for the construction of the patches in the enveloping lattice.

Definition at line 130 of file [LatticePatch.h](#).

### 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 LatticePatch()

LatticePatch::LatticePatch ( )

constructor setting up a default first lattice patch

Construct the lattice patch.

Definition at line 71 of file [LatticePatch.cpp](#).

```
00071   {
00072     // set default origin coordinates to (0,0,0)
00073     x0 = y0 = z0 = 0;
00074     // set default position in Lattice-Patchwork to (0,0,0)
00075     Llx = Lly = Llz = 0;
00076     // set default physical length for lattice patch to (0,0,0)
00077     lx = ly = lz = 0;
00078     // set default discrete length for lattice patch to (0,1,1)
00079     /* This is done in this manner as even in 1D simulations require a 1 point
00080      * width */
00081     nx = 0;
00082     ny = nz = 1;
00083
00084     // u is not initialized as it wouldn't make any sense before the dimensions
00085     // are set idem for the enveloping lattice
00086
00087     // set default statusFlags to non set
00088     statusFlags = 0;
00089 }
```

References [Llx](#), [Lly](#), [Llz](#), [lx](#), [ly](#), [lz](#), [nx](#), [ny](#), [nz](#), [statusFlags](#), [x0](#), [y0](#), and [z0](#).

### 5.9.2.2 ~LatticePatch()

LatticePatch::~LatticePatch ( )

destructor freeing parallel vectors

Destruct the patch and thereby destroy the NVectors.

Definition at line 92 of file [LatticePatch.cpp](#).

```
00092   {
00093     // Deallocate memory for solution vector
00094     if (statusFlags & FLatticePatchSetUp) {
00095       // Destroy data vectors
00096       #if defined(_OPENMP)
00097         N_VDestroy(u);
00098         N_VDestroy(du);
00099         N_VDestroy_OpenMP(uLocal);
00100         N_VDestroy_OpenMP(duLocal);
00101     #else
00102       N_VDestroy_Parallel(u);
00103       N_VDestroy_Parallel(du);
00104     #endif
00105   }
00106 }
```

References [du](#), [duLocal](#), [FLatticePatchSetUp](#), [statusFlags](#), [u](#), and [uLocal](#).

## 5.9.3 Member Function Documentation

### 5.9.3.1 checkFlag()

```
void LatticePatch::checkFlag (
    unsigned int flag ) const
```

function to check if a flag has been set and if not abort

Check if all flags are set.

Definition at line 631 of file [LatticePatch.cpp](#).

```
00631
00632     if (!(statusFlags & flag)) {
00633         std::string errorMessage;
00634         switch (flag) {
00635             case FLatticePatchSetUp:
00636                 errorMessage = "The Lattice patch was not set up please make sure to "
00637                             "initialize a Lattice topology";
00638                 break;
00639             case TranslocationLookupSetUp:
00640                 errorMessage = "The translocation lookup tables have not been generated, "
00641                             "please be sure to run generateTranslocationLookup()";
00642                 break;
00643             case GhostLayersInitialized:
00644                 errorMessage = "The space for the ghost layers has not been allocated, "
00645                             "please be sure that the ghost cells are initialized ";
00646                 break;
00647             case BuffersInitialized:
00648                 errorMessage = "The space for the buffers has not been allocated, please "
00649                             "be sure to run initializeBuffers()";
00650                 break;
00651         default:
00652             errorMessage = "Uppss, you've made a non-standard error, sadly I can't "
00653                             "help you there";
00654             break;
00655         }
00656         errorKill(errorMessage);
00657     }
00658     return;
00659 }
```

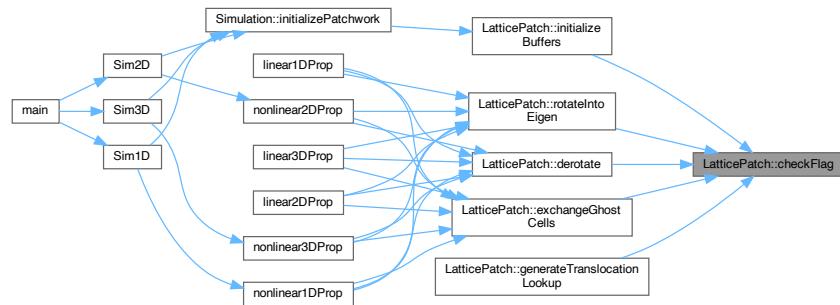
References [BuffersInitialized](#), [errorKill\(\)](#), [FLatticePatchSetUp](#), [GhostLayersInitialized](#), [statusFlags](#), and [TranslocationLookupSetUp](#).

Referenced by [derotate\(\)](#), [exchangeGhostCells\(\)](#), [generateTranslocationLookup\(\)](#), [initializeBuffers\(\)](#), and [rotateIntoEigen\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.2 `derive()`

```
void LatticePatch::derive (
```

function to derive the centered values in uAux and save them noncentered

Calculate derivatives in the patch ( $u_{\text{Aux}}$ ) in the specified direction.

Definition at line 662 of file [LatticePatch.cpp](#).

```

00662                                     {
00663     // ghost layer width adjusted to the chosen stencil order
00664     const int gLW = envelopeLattice->get_ghostLayerWidth();
00665     // dimensionality of data points -> 6
00666     const int dPD = envelopeLattice->get_dataPointDimension();
00667     // total width of patch in given direction including ghost layers at ends
00668     const sunindextype dirWidth = discreteSize(dir) + 2 * gLW;
00669     // width of patch only in given direction
00670     const sunindextype dirWidthO = discreteSize(dir);
00671     // size of plane perpendicular to given dimension
00672     const sunindextype perpPlainSize = discreteSize() / discreteSize(dir);
00673     // physical distance between points in that direction
00674     sunrealtype dxi = nan("0x12345");
00675     switch (dir) {
00676         case 1:
00677             dxi = dx;
00678             break;
00679         case 2:
00680             dxi = dy;
00681             break;
00682         case 3:
00683             dxi = dz;
00684             break;
00685     default:
00686         dxi = 1;
00687         errorKill("Tried to derive in the wrong direction");
00688         break;
00689     }
00690     // Derive according to chosen stencil accuracy order
00691     const int order = envelopeLattice->get_stencilOrder();
00692     switch (order) {
00693         case 1: // gLW=1
00694             #pragma omp parallel for default(none) \
00695             shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00696             for (sunindextype i = 0; i < perpPlainSize; i++) {
00697                 #pragma omp simd
00698                 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00699                     j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00700                     uAux[j + 0 - gLW * dPD] = s1b(&uAux[j + 0]) / dxi;
00701                     uAux[j + 1 - gLW * dPD] = s1b(&uAux[j + 1]) / dxi;
00702                     uAux[j + 2 - gLW * dPD] = s1f(&uAux[j + 2]) / dxi;

```

```

00703     uAux[j + 3 - gLW * dPD] = s1f(&uAux[j + 3]) / dxii;
00704     uAux[j + 4 - gLW * dPD] = s1f(&uAux[j + 4]) / dxii;
00705     uAux[j + 5 - gLW * dPD] = s1f(&uAux[j + 5]) / dxii;
00706 }
00707 }
00708 break;
00709 case 2: // gLW=2
00710 #pragma omp parallel for default(none) \
00711 shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00712 for (sunindextype i = 0; i < perpPlainSize; i++) {
00713 #pragma omp simd
00714 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00715     j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00716     uAux[j + 0 - gLW * dPD] = s2b(&uAux[j + 0]) / dxii;
00717     uAux[j + 1 - gLW * dPD] = s2b(&uAux[j + 1]) / dxii;
00718     uAux[j + 2 - gLW * dPD] = s2f(&uAux[j + 2]) / dxii;
00719     uAux[j + 3 - gLW * dPD] = s2f(&uAux[j + 3]) / dxii;
00720     uAux[j + 4 - gLW * dPD] = s2c(&uAux[j + 4]) / dxii;
00721     uAux[j + 5 - gLW * dPD] = s2c(&uAux[j + 5]) / dxii;
00722 }
00723 }
00724 break;
00725 case 3: // gLW=2
00726 #pragma omp parallel for default(none) \
00727 shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00728 for (sunindextype i = 0; i < perpPlainSize; i++) {
00729 #pragma omp simd
00730 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00731     j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00732     uAux[j + 0 - gLW * dPD] = s3b(&uAux[j + 0]) / dxii;
00733     uAux[j + 1 - gLW * dPD] = s3b(&uAux[j + 1]) / dxii;
00734     uAux[j + 2 - gLW * dPD] = s3f(&uAux[j + 2]) / dxii;
00735     uAux[j + 3 - gLW * dPD] = s3f(&uAux[j + 3]) / dxii;
00736     uAux[j + 4 - gLW * dPD] = s3f(&uAux[j + 4]) / dxii;
00737     uAux[j + 5 - gLW * dPD] = s3f(&uAux[j + 5]) / dxii;
00738 }
00739 }
00740 break;
00741 case 4: // gLW=3
00742 #pragma omp parallel for default(none) \
00743 shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00744 for (sunindextype i = 0; i < perpPlainSize; i++) {
00745 #pragma omp simd
00746 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00747     j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00748     uAux[j + 0 - gLW * dPD] = s4b(&uAux[j + 0]) / dxii;
00749     uAux[j + 1 - gLW * dPD] = s4b(&uAux[j + 1]) / dxii;
00750     uAux[j + 2 - gLW * dPD] = s4f(&uAux[j + 2]) / dxii;
00751     uAux[j + 3 - gLW * dPD] = s4f(&uAux[j + 3]) / dxii;
00752     uAux[j + 4 - gLW * dPD] = s4c(&uAux[j + 4]) / dxii;
00753     uAux[j + 5 - gLW * dPD] = s4c(&uAux[j + 5]) / dxii;
00754 }
00755 }
00756 break;
00757 case 5: // gLW=3
00758 #pragma omp parallel for default(none) \
00759 shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00760 for (sunindextype i = 0; i < perpPlainSize; i++) {
00761 #pragma omp simd
00762 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00763     j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00764     uAux[j + 0 - gLW * dPD] = s5b(&uAux[j + 0]) / dxii;
00765     uAux[j + 1 - gLW * dPD] = s5b(&uAux[j + 1]) / dxii;
00766     uAux[j + 2 - gLW * dPD] = s5f(&uAux[j + 2]) / dxii;
00767     uAux[j + 3 - gLW * dPD] = s5f(&uAux[j + 3]) / dxii;
00768     uAux[j + 4 - gLW * dPD] = s5f(&uAux[j + 4]) / dxii;
00769     uAux[j + 5 - gLW * dPD] = s5f(&uAux[j + 5]) / dxii;
00770 }
00771 }
00772 break;
00773 case 6: // gLW=4
00774 #pragma omp parallel for default(none) \
00775 shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00776 for (sunindextype i = 0; i < perpPlainSize; i++) {
00777 #pragma omp simd
00778 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00779     j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00780     uAux[j + 0 - gLW * dPD] = s6b(&uAux[j + 0]) / dxii;
00781     uAux[j + 1 - gLW * dPD] = s6b(&uAux[j + 1]) / dxii;
00782     uAux[j + 2 - gLW * dPD] = s6f(&uAux[j + 2]) / dxii;
00783     uAux[j + 3 - gLW * dPD] = s6f(&uAux[j + 3]) / dxii;
00784     uAux[j + 4 - gLW * dPD] = s6c(&uAux[j + 4]) / dxii;
00785     uAux[j + 5 - gLW * dPD] = s6c(&uAux[j + 5]) / dxii;
00786 }
00787 }
00788 break;
00789 case 7: // gLW=4

```

```

00790 #pragma omp parallel for default(none) \
00791 shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00792 for (sunindextype i = 0; i < perpPlainSize; i++) {
00793     #pragma omp simd
00794     for (sunindextype j = (i * dirWidth + gLW) * dPD;
00795         j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00796         uAux[j + 0 - gLW * dPD] = s7b(&uAux[j + 0]) / dxi;
00797         uAux[j + 1 - gLW * dPD] = s7b(&uAux[j + 1]) / dxi;
00798         uAux[j + 2 - gLW * dPD] = s7f(&uAux[j + 2]) / dxi;
00799         uAux[j + 3 - gLW * dPD] = s7f(&uAux[j + 3]) / dxi;
00800         uAux[j + 4 - gLW * dPD] = s7f(&uAux[j + 4]) / dxi;
00801         uAux[j + 5 - gLW * dPD] = s7f(&uAux[j + 5]) / dxi;
00802     }
00803 }
00804 break;
00805 case 8: // gLW=5
00806 #pragma omp parallel for default(none) \
00807 shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00808 for (sunindextype i = 0; i < perpPlainSize; i++) {
00809     #pragma omp simd
00810     for (sunindextype j = (i * dirWidth + gLW) * dPD;
00811         j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00812         uAux[j + 0 - gLW * dPD] = s8b(&uAux[j + 0]) / dxi;
00813         uAux[j + 1 - gLW * dPD] = s8b(&uAux[j + 1]) / dxi;
00814         uAux[j + 2 - gLW * dPD] = s8f(&uAux[j + 2]) / dxi;
00815         uAux[j + 3 - gLW * dPD] = s8f(&uAux[j + 3]) / dxi;
00816         uAux[j + 4 - gLW * dPD] = s8c(&uAux[j + 4]) / dxi;
00817         uAux[j + 5 - gLW * dPD] = s8c(&uAux[j + 5]) / dxi;
00818     }
00819 }
00820 break;
00821 case 9: // gLW=5
00822 #pragma omp parallel for default(none) \
00823 shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00824 for (sunindextype i = 0; i < perpPlainSize; i++) {
00825     #pragma omp simd
00826     for (sunindextype j = (i * dirWidth + gLW) * dPD;
00827         j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00828         uAux[j + 0 - gLW * dPD] = s9b(&uAux[j + 0]) / dxi;
00829         uAux[j + 1 - gLW * dPD] = s9b(&uAux[j + 1]) / dxi;
00830         uAux[j + 2 - gLW * dPD] = s9f(&uAux[j + 2]) / dxi;
00831         uAux[j + 3 - gLW * dPD] = s9f(&uAux[j + 3]) / dxi;
00832         uAux[j + 4 - gLW * dPD] = s9f(&uAux[j + 4]) / dxi;
00833         uAux[j + 5 - gLW * dPD] = s9f(&uAux[j + 5]) / dxi;
00834     }
00835 }
00836 break;
00837 case 10: // gLW=6
00838 #pragma omp parallel for default(none) \
00839 shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00840 for (sunindextype i = 0; i < perpPlainSize; i++) {
00841     #pragma omp simd
00842     for (sunindextype j = (i * dirWidth + gLW) * dPD;
00843         j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00844         uAux[j + 0 - gLW * dPD] = s10b(&uAux[j + 0]) / dxi;
00845         uAux[j + 1 - gLW * dPD] = s10b(&uAux[j + 1]) / dxi;
00846         uAux[j + 2 - gLW * dPD] = s10f(&uAux[j + 2]) / dxi;
00847         uAux[j + 3 - gLW * dPD] = s10f(&uAux[j + 3]) / dxi;
00848         uAux[j + 4 - gLW * dPD] = s10c(&uAux[j + 4]) / dxi;
00849         uAux[j + 5 - gLW * dPD] = s10c(&uAux[j + 5]) / dxi;
00850     }
00851 }
00852 break;
00853 case 11: // gLW=6
00854 #pragma omp parallel for default(none) \
00855 shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00856 for (sunindextype i = 0; i < perpPlainSize; i++) {
00857     #pragma omp simd
00858     for (sunindextype j = (i * dirWidth + gLW) * dPD;
00859         j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00860         uAux[j + 0 - gLW * dPD] = s11b(&uAux[j + 0]) / dxi;
00861         uAux[j + 1 - gLW * dPD] = s11b(&uAux[j + 1]) / dxi;
00862         uAux[j + 2 - gLW * dPD] = s11f(&uAux[j + 2]) / dxi;
00863         uAux[j + 3 - gLW * dPD] = s11f(&uAux[j + 3]) / dxi;
00864         uAux[j + 4 - gLW * dPD] = s11f(&uAux[j + 4]) / dxi;
00865         uAux[j + 5 - gLW * dPD] = s11f(&uAux[j + 5]) / dxi;
00866     }
00867 }
00868 break;
00869 case 12: // gLW=7
00870 #pragma omp parallel for default(none) \
00871 shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00872 for (sunindextype i = 0; i < perpPlainSize; i++) {
00873     #pragma omp simd
00874     for (sunindextype j = (i * dirWidth + gLW) * dPD;
00875         j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00876         uAux[j + 0 - gLW * dPD] = s12b(&uAux[j + 0]) / dxi;

```

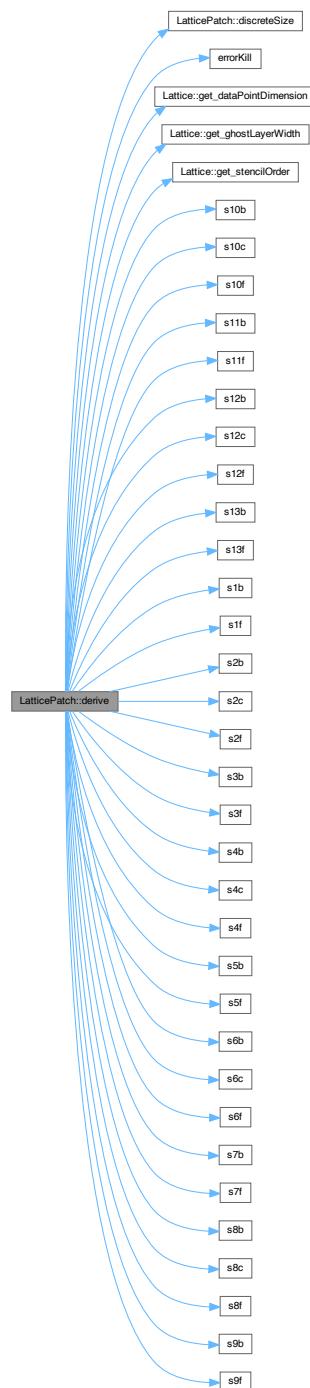
```

00877     uAux[j + 1 - gLW * dPD] = s12b(&uAux[j + 1]) / dx;
00878     uAux[j + 2 - gLW * dPD] = s12f(&uAux[j + 2]) / dx;
00879     uAux[j + 3 - gLW * dPD] = s12f(&uAux[j + 3]) / dx;
00880     uAux[j + 4 - gLW * dPD] = s12c(&uAux[j + 4]) / dx;
00881     uAux[j + 5 - gLW * dPD] = s12c(&uAux[j + 5]) / dx;
00882 }
00883 }
00884 break;
00885 case 13: // gLW=7
00886     // For all points in the plane perpendicular to the given direction
00887     #pragma omp parallel for default(none) \
00888     shared(perpPlainSize, dx, dirWidth, dirWidthO, gLW, dPD, uAux)
00889     for (sunindextype i = 0; i < perpPlainSize; i++) {
00890         // iterate through the derivation direction
00891         #pragma omp simd
00892         for (sunindextype j = (i * dirWidth
00893             + gLW /*to shift left by gLW below */) * dPD;
00894             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00895             // Compute the stencil derivative for any of the six field components
00896             // and update position by ghost width shift
00897             uAux[j + 0 - gLW * dPD] = s13b(&uAux[j + 0]) / dx;
00898             uAux[j + 1 - gLW * dPD] = s13b(&uAux[j + 1]) / dx;
00899             uAux[j + 2 - gLW * dPD] = s13f(&uAux[j + 2]) / dx;
00900             uAux[j + 3 - gLW * dPD] = s13f(&uAux[j + 3]) / dx;
00901             uAux[j + 4 - gLW * dPD] = s13f(&uAux[j + 4]) / dx;
00902             uAux[j + 5 - gLW * dPD] = s13f(&uAux[j + 5]) / dx;
00903         }
00904     }
00905     break;
00906 default:
00907     errorKill("Please set an existing stencil order");
00908     break;
00909 }
00910 }
00911 }
```

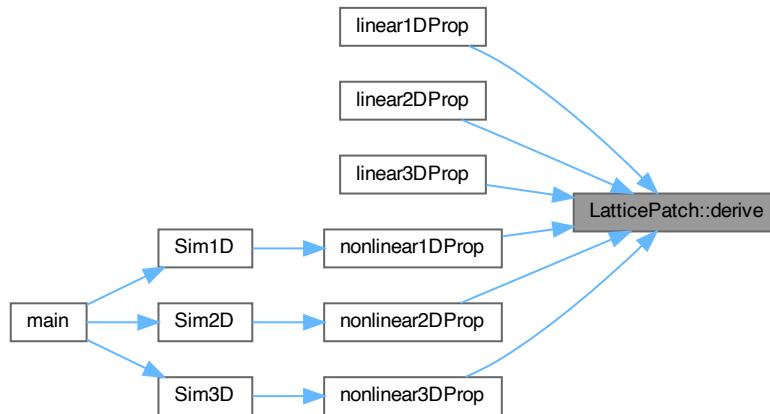
References [discreteSize\(\)](#), [dx](#), [dy](#), [dz](#), [envelopeLattice](#), [errorKill\(\)](#), [Lattice::get\\_dataPointDimension\(\)](#), [Lattice::get\\_ghostLayerWidth\(\)](#), [Lattice::get\\_stencilOrder\(\)](#), [s10b\(\)](#), [s10c\(\)](#), [s10f\(\)](#), [s11b\(\)](#), [s11f\(\)](#), [s12b\(\)](#), [s12c\(\)](#), [s12f\(\)](#), [s13b\(\)](#), [s13f\(\)](#), [s1b\(\)](#), [s1f\(\)](#), [s2b\(\)](#), [s2c\(\)](#), [s2f\(\)](#), [s3b\(\)](#), [s3f\(\)](#), [s4b\(\)](#), [s4c\(\)](#), [s4f\(\)](#), [s5b\(\)](#), [s5f\(\)](#), [s6b\(\)](#), [s6c\(\)](#), [s6f\(\)](#), [s7b\(\)](#), [s7f\(\)](#), [s8b\(\)](#), [s8c\(\)](#), [s8f\(\)](#), [s9b\(\)](#), [s9f\(\)](#), and [uAux](#).

Referenced by [linear1DProp\(\)](#), [linear2DProp\(\)](#), [linear3DProp\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), and [nonlinear3DProp\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.3 derotate()

```
void LatticePatch::derotate (
    int dir,
    sunrealtype * buffOut )
```

function to derotate uAux into dudata lattice direction of x

Derotate uAux with transposed rotation matrices and write to derivative buffer – normalization is done here by the factor 1/2

Definition at line 446 of file [LatticePatch.cpp](#).

```
00446 // Check that the lattice as well as the translocation lookups have been set
00447 // up;
00448 checkFlag(FLatticePatchSetUp);
00449 checkFlag(TranslocationLookupSetUp);
00450 const int dPD = envelopeLattice->get_dataPointDimension();
00451 const int gLW = envelopeLattice->get_ghostLayerWidth();
00452 const sunindextype totalNP = discreteSize();
00453 sunindextype ii = 0, target = 0;
00454 switch (dir) {
00455 case 1:
00456 #pragma omp parallel for simd \
00457 private(ii, target) \
00458 shared(dPD, gLW, totalNP, uTox, uAux, buffOut) \
00459 schedule(static)
00460 for (sunindextype i = 0; i < totalNP; i++) {
00461     // get correct indices in u and rotation space
00462     target = dPD * i;
00463     ii = dPD * (uTox[i] - gLW);
00464     buffOut[target + 0] = uAux[5 + ii];
00465     buffOut[target + 1] = (-uAux[ii] + uAux[2 + ii]) / 2.;
00466     buffOut[target + 2] = (uAux[1 + ii] - uAux[3 + ii]) / 2.;
00467     buffOut[target + 3] = uAux[4 + ii];
00468     buffOut[target + 4] = (uAux[1 + ii] + uAux[3 + ii]) / 2.;
00469     buffOut[target + 5] = (uAux[ii] + uAux[2 + ii]) / 2.;
00470 }
00471 break;
00472 case 2:
00473 #pragma omp parallel for simd \
00474 private(ii, target) \
```

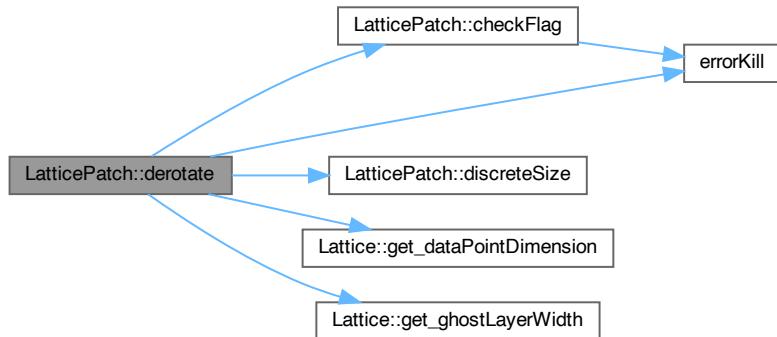
```

00476     shared(dPD, gLW, totalNP, uTox, uAux, buffOut) \
00477     schedule(static)
00478     for (sunindextype i = 0; i < totalNP; i++) {
00479         target = dPD * i;
00480         ii = dPD * (uToy[i] - gLW);
00481         buffOut[target + 0] = (uAux[ii] - uAux[2 + ii]) / 2.;
00482         buffOut[target + 1] = uAux[5 + ii];
00483         buffOut[target + 2] = (-uAux[1 + ii] + uAux[3 + ii]) / 2.;
00484         buffOut[target + 3] = (uAux[1 + ii] + uAux[3 + ii]) / 2.;
00485         buffOut[target + 4] = uAux[4 + ii];
00486         buffOut[target + 5] = (uAux[ii] + uAux[2 + ii]) / 2.;
00487     }
00488     break;
00489 case 3:
00490     #pragma omp parallel for simd \
00491     private(ii, target) \
00492     shared(dPD, gLW, totalNP, uTox, uAux, buffOut) \
00493     schedule(static)
00494     for (sunindextype i = 0; i < totalNP; i++) {
00495         target = dPD * i;
00496         ii = dPD * (uToz[i] - gLW);
00497         buffOut[target + 0] = (-uAux[ii] + uAux[2 + ii]) / 2.;
00498         buffOut[target + 1] = (uAux[1 + ii] - uAux[3 + ii]) / 2.;
00499         buffOut[target + 2] = uAux[5 + ii];
00500         buffOut[target + 3] = (uAux[1 + ii] + uAux[3 + ii]) / 2.;
00501         buffOut[target + 4] = (uAux[ii] + uAux[2 + ii]) / 2.;
00502         buffOut[target + 5] = uAux[4 + ii];
00503     }
00504     break;
00505 default:
00506     errorKill("Tried to derotate from the wrong direction");
00507     break;
00508 }
00509 }
```

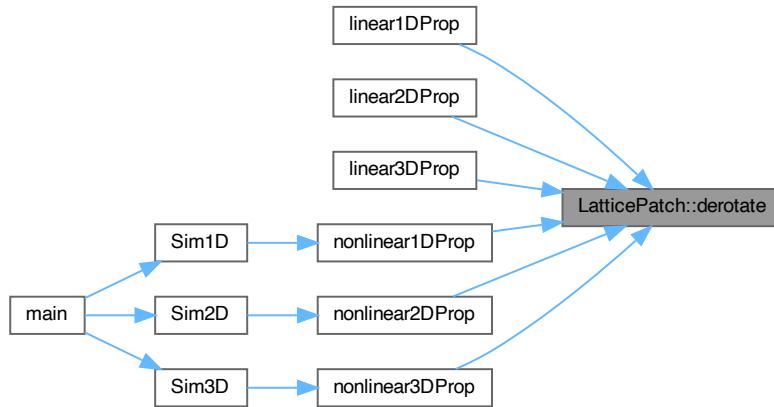
References [checkFlag\(\)](#), [discreteSize\(\)](#), [envelopeLattice](#), [errorKill\(\)](#), [FLatticePatchSetUp](#), [Lattice::get\\_dataPointDimension\(\)](#), [Lattice::get\\_ghostLayerWidth\(\)](#), [TranslocationLookupSetUp](#), [uAux](#), [uTox](#), [uToy](#), and [uToz](#).

Referenced by [linear1DProp\(\)](#), [linear2DProp\(\)](#), [linear3DProp\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), and [nonlinear3DProp\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.9.3.4 discreteSize()

```
sunindextype LatticePatch::discreteSize (
    int dir = 0 ) const
```

function to get the discrete size of the [LatticePatch](#)

Return the discrete size of the patch: number of lattice patch points in specified dimension

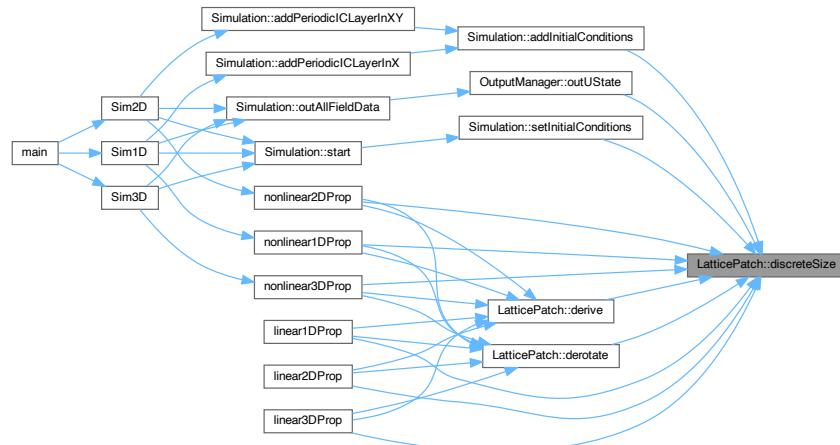
Definition at line 203 of file [LatticePatch.cpp](#).

```
00203
00204     switch (dir) {
00205         case 0:
00206             return nx * ny * nz;
00207         case 1:
00208             return nx;
00209         case 2:
00210             return ny;
00211         case 3:
00212             return nz;
00213     // case 4: return uAux.size(); // for debugging
00214     default:
00215         return -1;
00216     }
00217 }
```

References [nx](#), [ny](#), and [nz](#).

Referenced by [Simulation::addInitialConditions\(\)](#), [derive\(\)](#), [derotate\(\)](#), [linear1DProp\(\)](#), [linear2DProp\(\)](#), [linear3DProp\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), [nonlinear3DProp\(\)](#), [OutputManager::outUState\(\)](#), and [Simulation::setInitialConditions\(\)](#).

Here is the caller graph for this function:



### 5.9.3.5 exchangeGhostCells()

```
void LatticePatch::exchangeGhostCells (
    const int dir )
```

function to exchange ghost cells

Perform the ghost cell exchange in a specified direction.

Definition at line 527 of file [LatticePatch.cpp](#).

```
00527
00528 // Check that the lattice has been set up
00529 checkFlag(FLatticeDimensionSet);
00530 checkFlag(FLatticePatchSetUp);
00531 // Variables to per dimension calculate the halo indices, and distance to
00532 // other side halo boundary
00533 int mx = 1, my = 1, mz = 1, distToRight = 1;
00534 const int gLW = envelopeLattice->get_ghostLayerWidth();
00535 // In the chosen direction m is set to ghost layer width while the others
00536 // remain to form the plane
00537 switch (dir) {
00538 case 1:
00539     mx = gLW;
00540     my = ny;
00541     mz = nz;
00542     distToRight = (nx - gLW);
00543     break;
00544 case 2:
00545     mx = nx;
00546     my = gLW;
00547     mz = nz;
00548     distToRight = nx * (ny - gLW);
00549     break;
00550 case 3:
00551     mx = nx;
00552     my = ny;
00553     mz = gLW;
00554     distToRight = nx * ny * (nz - gLW);
00555     break;
00556 }
00557 // total number of exchanged points
00558 const int dPD = envelopeLattice->get_dataPointDimension();
00559 const sunindextype exchangeSize = mx * my * mz * dPD;
00560 // provide size of the halos for ghost cells
```

```

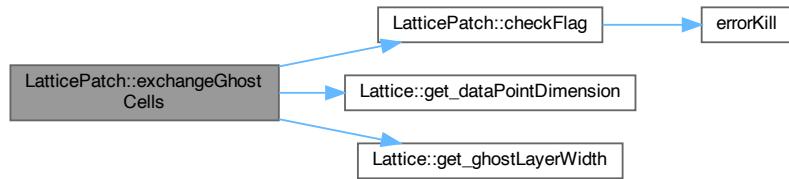
00561     ghostCellLeft.resize(exchangeSize);
00562     ghostCellRight.resize(ghostCellLeft.size());
00563     ghostCellLeftToSend.resize(ghostCellLeft.size());
00564     ghostCellRightToSend.resize(ghostCellLeft.size());
00565     gCLData = &ghostCellLeft[0];
00566     gCRData = &ghostCellRight[0];
00567     statusFlags |= GhostLayersInitialized;
00568
00569     // Initialize running index li for the halo buffers, and index ui of uData for
00570     // data transfer
00571     sunindextype li = 0, ui = 0;
00572     // Fill the halo buffers
00573     #pragma omp parallel for default(none) \
00574     private(ui, li) \
00575     shared(nx, ny, mx, my, mz, dPD, distToRight, uData, \
00576             ghostCellLeftToSend, ghostCellRightToSend)
00577     for (sunindextype iz = 0; iz < mz; iz++) {
00578         for (sunindextype iy = 0; iy < my; iy++) {
00579             // uData vector start index of halo data to be transferred
00580             // with each z-step add the whole xy-plane and with y-step the x-range ->
00581             // iterate all x-ranges
00582             ui = (iz * nx * ny + iy * nx) * dPD;
00583             // increase halo index by transferred items of previous iteration steps
00584             li = (iz * my * mx + iy * mx) * dPD;
00585             // copy left halo data from uData to buffer, transfer size is given by
00586             // x-length (not x-range)
00587             std::copy(&uData[ui], &uData[ui + mx * dPD], &ghostCellLeftToSend[li]);
00588             ui += distToRight * dPD;
00589             std::copy(&uData[ui], &uData[ui + mx * dPD], &ghostCellRightToSend[li]);
00590     }
00591 }
00592
00593 /* Send and receive the data to and from neighboring latticePatches */
00594 // Adjust direction to cartesian communicator
00595 int dim = 2; // default for dir==1
00596 if (dir == 2) {
00597     dim = 1;
00598 } else if (dir == 3) {
00599     dim = 0;
00600 }
00601 int rank_source = 0, rank_dest = 0;
00602 MPI_Cart_shift(envelopeLattice->comm, dim, -1, &rank_source,
00603                 &rank_dest); // s.t. rank_dest is left & v.v.
00604
00605 // nonblocking Irecv/Isend
00606
00607 MPI_Request requests[4];
00608 MPI_Irecv(&ghostCellRight[0], exchangeSize, MPI_SUNREALTYPE, rank_source, 1,
00609 envelopeLattice->comm, &requests[0]);
00610 MPI_Isend(&ghostCellLeftToSend[0], exchangeSize, MPI_SUNREALTYPE, rank_dest,
00611 1, envelopeLattice->comm, &requests[1]);
00612 MPI_Irecv(&ghostCellLeft[0], exchangeSize, MPI_SUNREALTYPE, rank_dest, 2,
00613 envelopeLattice->comm, &requests[2]);
00614 MPI_Isend(&ghostCellRightToSend[0], exchangeSize, MPI_SUNREALTYPE,
00615 rank_source, 2, envelopeLattice->comm, &requests[3]);
00616 MPI_Waitall(4, requests, MPI_STATUS_IGNORE);
00617
00618 // blocking Sendrecv:
00619 /*
00620 MPI_Sendrecv(&ghostCellLeftToSend[0], exchangeSize, MPI_SUNREALTYPE,
00621             rank_dest, 1, &ghostCellRight[0], exchangeSize, MPI_SUNREALTYPE,
00622             rank_source, 1, envelopeLattice->comm, MPI_STATUS_IGNORE);
00623 MPI_Sendrecv(&ghostCellRightToSend[0], exchangeSize, MPI_SUNREALTYPE,
00624             rank_source, 2, &ghostCellLeft[0], exchangeSize, MPI_SUNREALTYPE,
00625             rank_dest, 2, envelopeLattice->comm, MPI_STATUS_IGNORE);
00626 */
00627 }
00628 }

```

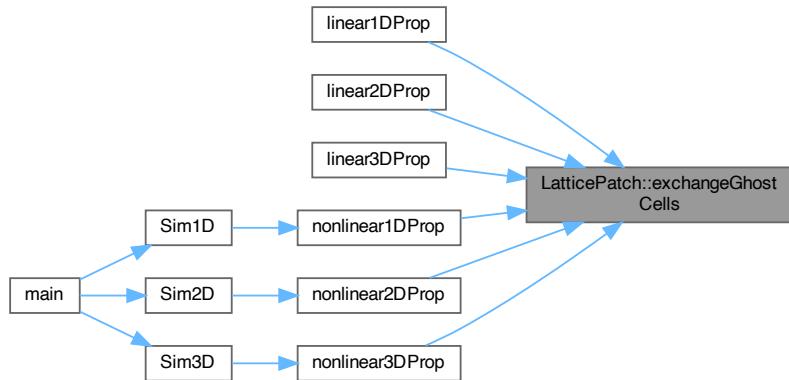
References [checkFlag\(\)](#), [Lattice::comm](#), [envelopeLattice](#), [FLatticeDimensionSet](#), [FLatticePatchSetUp](#), [gCLData](#), [gCRData](#), [Lattice::get\\_dataPointDimension\(\)](#), [Lattice::get\\_ghostLayerWidth\(\)](#), [ghostCellLeft](#), [ghostCellLeftToSend](#), [ghostCellRight](#), [ghostCellRightToSend](#), [GhostLayersInitialized](#), [nx](#), [ny](#), [nz](#), [statusFlags](#), and [uData](#).

Referenced by [linear1DProp\(\)](#), [linear2DProp\(\)](#), [linear3DProp\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), and [nonlinear3DProp\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.6 generateTranslocationLookup()

```
void LatticePatch::generateTranslocationLookup( )
```

function to fill out the lookup tables for cache efficiency

In order to avoid cache misses: create vectors to translate u vector into space coordinates and vice versa and same for left and right ghost layers to space

Definition at line 253 of file [LatticePatch.cpp](#).

```

00253
00254 // Check that the lattice has been set up
00255 checkFlag(FLatticeDimensionSet);
00256 // lengths for auxilliary layers, including ghost layers
00257 const int gLW = envelopeLattice->get_ghostLayerWidth();
00258 const sunindextype mx = nx + 2 * gLW;
00259 const sunindextype my = ny + 2 * gLW;
00260 const sunindextype mz = nz + 2 * gLW;
00261 // sizes for lookup vectors
00262 const sunindextype totalNP = nx * ny * nz;
00263 const sunindextype haloXSize = mx * ny * nz;
00264 const sunindextype haloYSIZE = nx * my * nz;
00265 const sunindextype haloZSize = nx * ny * mz;
  
```

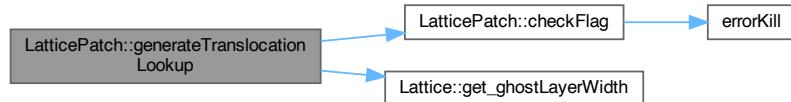
```

00266 // generate u->uAux
00267 uTox.resize(totalNP);
00268 uToy.resize(totalNP);
00269 uToz.resize(totalNP);
00270 // generate uAux->u with length including halo
00271 xTou.resize(haloXSize);
00272 yTou.resize(haloYSize);
00273 zTou.resize(haloZSize);
00274 // same for ghost layer lookup tables
00275 const sunindextype ghostXSize = gLW * ny * nz;
00276 const sunindextype ghostYSize = gLW * nx * nz;
00277 const sunindextype ghostZSize = gLW * nx * ny;
00278 lgcTox.resize(ghostXSize);
00279 rgcTox.resize(ghostXSize);
00280 lgcToy.resize(ghostYSize);
00281 rgcToy.resize(ghostYSize);
00282 lgcToz.resize(ghostZSize);
00283 rgcToz.resize(ghostZSize);
00284 // variables for cartesian position in the 3D discrete lattice
00285 sunindextype px = 0, py = 0, pz = 0;
00286 // Fill the lookup tables
00287 #pragma omp parallel default(none) \
00288 private(px, py, pz) \
00289 shared(uTox, uToy, uToz, xTou, yTou, zTou, \
00290         nx, ny, mx, my, mz, gLW, totalNP, \
00291         lgcTox, rgcTox, lgcToy, rgcToy, lgcToz, rgcToz, \
00292         ghostXSize, ghostYSize, ghostZSize)
00293 {
00294 #pragma omp for simd schedule(static)
00295 for (sunindextype i = 0; i < totalNP; i++) { // loop over the patch
00296     // calculate cartesian coordinates
00297     px = i % nx;
00298     py = (i / nx) % ny;
00299     pz = (i / nx) / ny;
00300     // fill lookups extended by halos (useful for y and z direction)
00301     uTox[i] = (px + gLW) + py * mx +
00302                 pz * mx * ny; // unroll (de-flatten) cartesian dimension
00303     xTou[px + py * mx + pz * mx * ny] =
00304         i; // match cartesian point to u location
00305     uToy[i] = (py + gLW) + pz * my + px * my * nz;
00306     yTou[py + pz * my + px * my * nz] = i;
00307     uToz[i] = (pz + gLW) + px * mz + py * mz * nx;
00308     zTou[pz + px * mz + py * mz * nx] = i;
00309 }
00310 #pragma omp for simd schedule(static)
00311 for (sunindextype i = 0; i < ghostXSize; i++) {
00312     px = i % gLW;
00313     py = (i / gLW) % ny;
00314     pz = (i / gLW) / ny;
00315     lgcTox[i] = px + py * mx + pz * mx * ny;
00316     rgcTox[i] = px + nx + gLW + py * mx + pz * mx * ny;
00317 }
00318 #pragma omp for simd schedule(static)
00319 for (sunindextype i = 0; i < ghostYSize; i++) {
00320     px = i % nx;
00321     py = (i / nx) % gLW;
00322     pz = (i / nx) / gLW;
00323     lgcToy[i] = py + pz * my + px * my * nz;
00324     rgcToy[i] = py + ny + gLW + pz * my + px * my * nz;
00325 }
00326 #pragma omp for simd schedule(static)
00327 for (sunindextype i = 0; i < ghostZSize; i++) {
00328     px = i % nx;
00329     py = (i / nx) % ny;
00330     pz = (i / nx) / ny;
00331     lgcToz[i] = pz + px * mz + py * mz * nx;
00332     rgcToz[i] = pz + nz + gLW + px * mz + py * mz * nx;
00333 }
00334 }
00335 statusFlags |= TranslocationLookupSetUp;
00336 }

```

References [checkFlag\(\)](#), [envelopeLattice](#), [FLatticeDimensionSet](#), [Lattice::get\\_ghostLayerWidth\(\)](#), [lgcTox](#), [lgcToy](#), [lgcToz](#), [nx](#), [ny](#), [nz](#), [rgcTox](#), [rgcToy](#), [rgcToz](#), [statusFlags](#), [TranslocationLookupSetUp](#), [uTox](#), [uToy](#), [uToz](#), [xTou](#), [yTou](#), and [zTou](#).

Here is the call graph for this function:



### 5.9.3.7 getDelta()

```
sunrealtype LatticePatch::getDelta (
    const int dir ) const
```

function to get distance between points

Return the distance between points in the patch in a dimension.

Definition at line 235 of file [LatticePatch.cpp](#).

```
00235
00236     switch (dir) {
00237         case 1:
00238             return dx;
00239         case 2:
00240             return dy;
00241         case 3:
00242             return dz;
00243     default:
00244         errorKill(
00245             "LatticePatch::getDelta function called with wrong dir parameter");
00246         return -1;
00247     }
00248 }
```

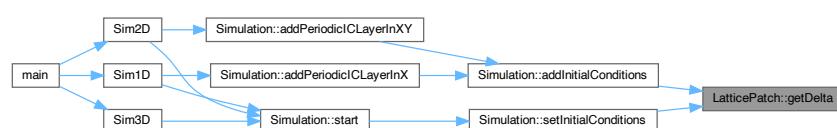
References [dx](#), [dy](#), [dz](#), and [errorKill\(\)](#).

Referenced by [Simulation::addInitialConditions\(\)](#), and [Simulation::setInitialConditions\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.8 initializeBuffers()

```
void LatticePatch::initializeBuffers ( )
```

initialize buffers to save derivatives

Create buffers to save derivative values, optimizing computational load.

Definition at line 512 of file [LatticePatch.cpp](#).

```
00512                                     {
00513     // Check that the lattice has been set up
00514     checkFlag(FLatticeDimensionSet);
00515     const int dPD = envelopeLattice->get_dataPointDimension();
00516     buffX.resize(nx * ny * nz * dPD);
00517     buffY.resize(nx * ny * nz * dPD);
00518     buffZ.resize(nx * ny * nz * dPD);
00519     // Set pointers used for propagation functions
00520     buffData[0] = &buffX[0];
00521     buffData[1] = &buffY[0];
00522     buffData[2] = &buffZ[0];
00523     statusFlags |= BuffersInitialized;
00524 }
```

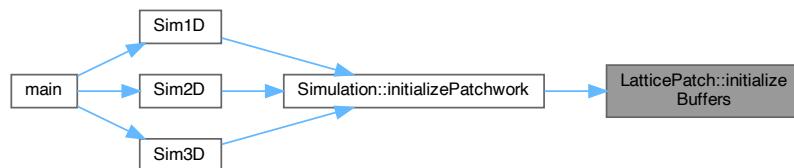
References [buffData](#), [BuffersInitialized](#), [buffX](#), [buffY](#), [buffZ](#), [checkFlag\(\)](#), [envelopeLattice](#), [FLatticeDimensionSet](#), [Lattice::get\\_dataPointDimension\(\)](#), [nx](#), [ny](#), [nz](#), and [statusFlags](#).

Referenced by [Simulation::initializePatchwork\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.9 origin()

```
sunrealtype LatticePatch::origin (
    const int dir ) const
```

function to get the origin of the patch

Return the physical origin of the patch in a dimension.

Definition at line 220 of file [LatticePatch.cpp](#).

```
00220
00221     switch (dir) {
00222     case 1:
00223         return x0;
00224     case 2:
00225         return y0;
00226     case 3:
00227         return z0;
00228     default:
00229         errorKill("LatticePatch::origin function called with wrong dir parameter");
00230         return -1;
00231     }
00232 }
```

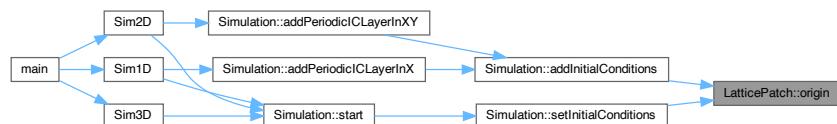
References [errorKill\(\)](#), [x0](#), [y0](#), and [z0](#).

Referenced by [Simulation::addInitialConditions\(\)](#), and [Simulation::setInitialConditions\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.10 rotateIntoEigen()

```
void LatticePatch::rotateIntoEigen (
    const int dir )
```

function to rotate u into Z-matrix eigenraum

Rotate into eigenraum along R matrices of paper using the rotation methods; uAuxData gets the rotated left-halo-, inner-patch-, right-halo-data

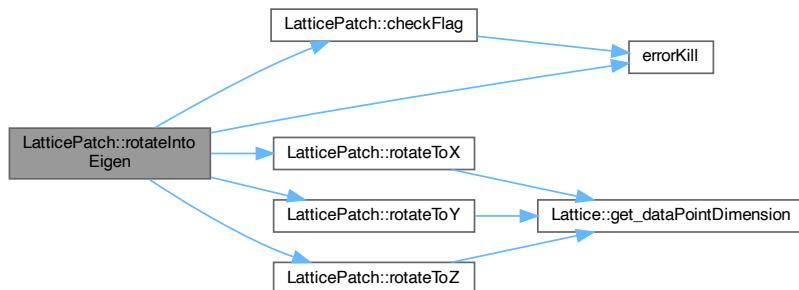
Definition at line 341 of file [LatticePatch.cpp](#).

```
00341
00342     // Check that the lattice, ghost layers as well as the translocation lookups
00343     // have been set up;
00344     checkFlag(FLatticePatchSetUp);
00345     checkFlag(TranslocationLookupSetUp);
00346     checkFlag(GhostLayersInitialized); // this check is only after call to
00347     // exchange ghost cells
00348     switch (dir) {
00349     case 1:
00350         rotateToX(uAuxData, gCLData, lgcTox);
00351         rotateToX(uAuxData, uData, uTox);
00352         rotateToX(uAuxData, gCRData, rgcTox);
00353         break;
00354     case 2:
00355         rotateToY(uAuxData, gCLData, lgcToy);
00356         rotateToY(uAuxData, uData, uToy);
00357         rotateToY(uAuxData, gCRData, rgcToy);
00358         break;
00359     case 3:
00360         rotateToZ(uAuxData, gCLData, lgcToz);
00361         rotateToZ(uAuxData, uData, uToz);
00362         rotateToZ(uAuxData, gCRData, rgcToz);
00363         break;
00364     default:
00365         errorKill("Tried to rotate into the wrong direction");
00366         break;
00367     }
00368 }
```

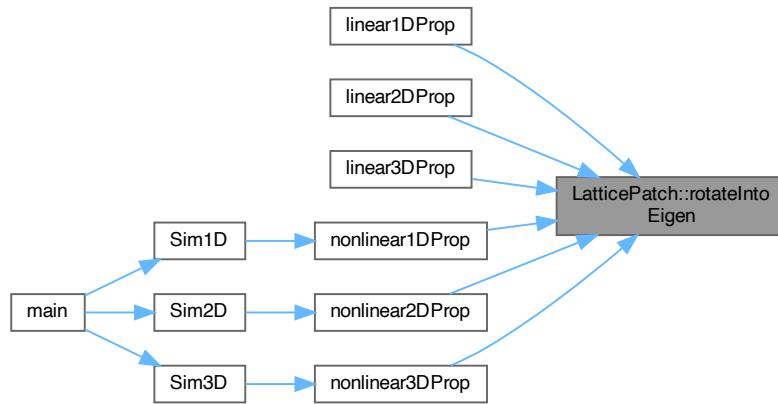
References [checkFlag\(\)](#), [errorKill\(\)](#), [FLatticePatchSetUp](#), [gCLData](#), [gCRData](#), [GhostLayersInitialized](#), [lgcTox](#), [lgcToy](#), [lgcToz](#), [rgcTox](#), [rgcToy](#), [rgcToz](#), [rotateToX\(\)](#), [rotateToY\(\)](#), [rotateToZ\(\)](#), [TranslocationLookupSetUp](#), [uAuxData](#), [uData](#), [uTox](#), [uToy](#), and [uToz](#).

Referenced by [linear1DProp\(\)](#), [linear2DProp\(\)](#), [linear3DProp\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), and [nonlinear3DProp\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.11 `rotateToX()`

```

void LatticePatch::rotateToX (
    sunrealtype * outArray,
    const sunrealtype * inArray,
    const std::vector< sunindextype > & lookup ) [inline], [private]
  
```

Rotate and translocate an input array according to a lookup into an output array

Rotate halo and inner-patch data vectors with rotation matrix Rx into eigenspace of Z matrix and write to auxiliary vector

Definition at line 372 of file [LatticePatch.cpp](#).

```

00374     sunindextype ii = 0, target = 0;
00375     const sunindextype size = lookup.size();
00376     const int dPD = envelopeLattice->get_dataPointDimension();
00377     #pragma omp parallel for simd \
00378     private(target, ii) \
00379     shared(lookup, outArray, inArray, size, dPD) \
00380     schedule(static)
00381     for (sunindextype i = 0; i < size; i++) {
00382         // get correct u-vector and spatial indices along previously defined lookup
00383         // tables
00384         target = dPD * lookup[i];
00385         ii = dPD * i;
00386         outArray[target + 0] = -inArray[1 + ii] + inArray[5 + ii];
00387         outArray[target + 1] = inArray[2 + ii] + inArray[4 + ii];
00388         outArray[target + 2] = inArray[1 + ii] + inArray[5 + ii];
00389         outArray[target + 3] = -inArray[2 + ii] + inArray[4 + ii];
00390         outArray[target + 4] = inArray[3 + ii];
00391         outArray[target + 5] = inArray[ii];
00392     }
00393 }
00394 }
```

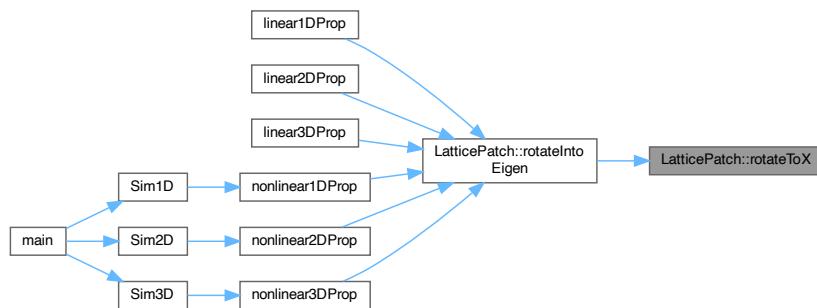
References `envelopeLattice`, and `Lattice::get_dataPointDimension()`.

Referenced by `rotateIntoEigen()`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.12 rotateToY()

```

void LatticePatch::rotateToY (
    sunrealtype * outArray,
    const sunrealtype * inArray,
    const std::vector< sunindextype > & lookup ) [inline], [private]
  
```

Rotate halo and inner-patch data vectors with rotation matrix Ry into eigenspace of Z matrix and write to auxiliary vector

Definition at line 398 of file [LatticePatch.cpp](#).

```

00400
00401 sunindextype ii = 0, target = 0;
00402 const int dPD = envelopeLattice->get_dataPointDimension();
00403 const sunindextype size = lookup.size();
00404 #pragma omp parallel for simd \
00405 private(target, ii) \
00406 shared(lookup, outArray, inArray, size, dPD) \
00407 schedule(static)
00408 for (sunindextype i = 0; i < size; i++) {
00409     target = dPD * lookup[i];
00410     ii = dPD * i;
00411     outArray[target + 0] = inArray[ii] + inArray[5 + ii];
00412     outArray[target + 1] = -inArray[2 + ii] + inArray[3 + ii];
00413     outArray[target + 2] = -inArray[ii] + inArray[5 + ii];
00414     outArray[target + 3] = inArray[2 + ii] + inArray[3 + ii];
00415     outArray[target + 4] = inArray[4 + ii];
00416     outArray[target + 5] = inArray[1 + ii];
00417 }
00418 }
  
```

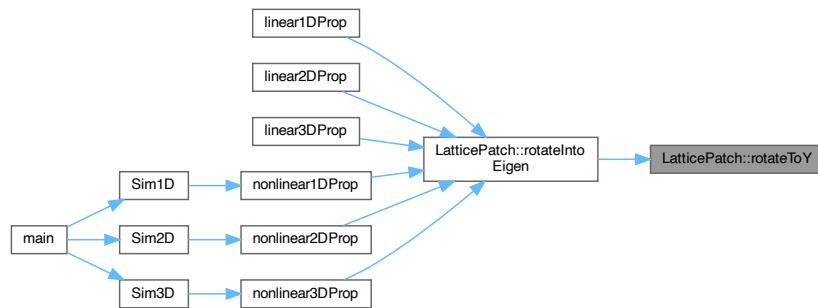
References [envelopeLattice](#), and [Lattice::get\\_dataPointDimension\(\)](#).

Referenced by [rotateIntoEigen\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.9.3.13 rotateToZ()

```

void LatticePatch::rotateToZ (
    sunrealtype * outArray,
    const sunrealtype * inArray,
    const std::vector< sunindextype > & lookup ) [inline], [private]
  
```

Rotate halo and inner-patch data vectors with rotation matrix Rz into eigenspace of Z matrix and write to auxiliary vector

Definition at line 422 of file [LatticePatch.cpp](#).

```

00424
00425     sunindextype ii = 0, target = 0;
00426     const sunindextype size = lookup.size();
00427     const int dPD = envelopeLattice->get_dataPointDimension();
00428     #pragma omp parallel for simd \
00429     private(target, ii) \
00430     shared(lookup, outArray, inArray, size, dPD) \
00431     schedule(static)
00432     for (sunindextype i = 0; i < size; i++) {
00433         target = dPD * lookup[i];
00434         ii = dPD * i;
00435         outArray[target + 0] = -inArray[ii] + inArray[4 + ii];
00436         outArray[target + 1] = inArray[1 + ii] + inArray[3 + ii];
00437         outArray[target + 2] = inArray[ii] + inArray[4 + ii];
  
```

```

00438     outArray[target + 3] = -inArray[1 + ii] + inArray[3 + ii];
00439     outArray[target + 4] = inArray[5 + ii];
00440     outArray[target + 5] = inArray[2 + ii];
00441 }
00442 }
```

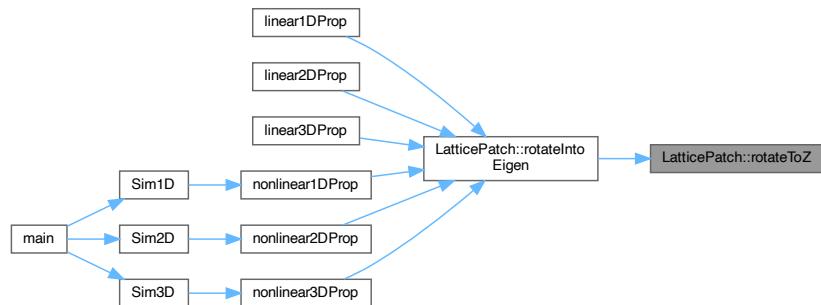
References [envelopeLattice](#), and [Lattice::get\\_dataPointDimension\(\)](#).

Referenced by [rotateIntoEigen\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.9.4 Friends And Related Function Documentation

### 5.9.4.1 generatePatchwork

```

int generatePatchwork (
    const Lattice & envelopeLattice,
    LatticePatch & patchToMold,
    const int DLx,
    const int DLy,
    const int DLz ) [friend]
```

friend function for creating the patchwork slicing of the overall lattice

Definition at line 109 of file [LatticePatch.cpp](#).

```
00111 {
```

```

0012 // Retrieve the ghost layer depth
0013 const int gLW = envelopeLattice.get_ghostLayerWidth();
0014 // Retrieve the data point dimension
0015 const int dPD = envelopeLattice.get_dataPointDimension();
0016 // MPI process/patch
0017 const int my_prc = envelopeLattice.my_prc;
0018 // Determine thickness of the slice
0019 const sunindextype tot_NOXP = envelopeLattice.get_tot_nx();
0020 const sunindextype tot_NOYP = envelopeLattice.get_tot_ny();
0021 const sunindextype tot_NOZP = envelopeLattice.get_tot_nz();
0022 // position of the patch in the lattice of patches -> process associated to
0023 // position
0024 const sunindextype LIx = my_prc % DLx;
0025 const sunindextype LIy = (my_prc / DLx) % DLy;
0026 const sunindextype LIz = (my_prc / DLx) / DLy;
0027 // Determine the number of points in the patch and first absolute points in
0028 // each dimension
0029 const sunindextype local_NOXP = tot_NOXP / DLx;
0030 const sunindextype local_NOYP = tot_NOYP / DLy;
0031 const sunindextype local_NOZP = tot_NOZP / DLz;
0032 // absolute positions of the first point in each dimension
0033 const sunindextype firstXPoint = local_NOXP * LIx;
0034 const sunindextype firstYPoint = local_NOYP * LIy;
0035 const sunindextype firstZPoint = local_NOZP * LIz;
0036 // total number of points in the patch
0037 const sunindextype local_NODP = dPD * local_NOXP * local_NOYP * local_NOZP;
0038
0039 // Set patch up with above derived quantities
0040 patchToMold.dx = envelopeLattice.get_dx();
0041 patchToMold.dy = envelopeLattice.get_dy();
0042 patchToMold.dz = envelopeLattice.get_dz();
0043 patchToMold.x0 = firstXPoint * patchToMold.dx;
0044 patchToMold.y0 = firstYPoint * patchToMold.dy;
0045 patchToMold.z0 = firstZPoint * patchToMold.dz;
0046 patchToMold.LIx = LIx;
0047 patchToMold.LIy = LIy;
0048 patchToMold.LIz = LIz;
0049 patchToMold.nx = local_NOXP;
0050 patchToMold.ny = local_NOYP;
0051 patchToMold.nz = local_NOZP;
0052 patchToMold.lx = patchToMold.nx * patchToMold.dx;
0053 patchToMold.ly = patchToMold.ny * patchToMold.dy;
0054 patchToMold.lz = patchToMold.nz * patchToMold.dz;
0055
0056 #ifdef _OPENMP
0057 // OpenMP and MPI+X NVectors interoperability
0058 // OpenMP NVectors with local patch size
0059 int num_threads = 1;
0060 num_threads = omp_get_max_threads();
0061 patchToMold.uLocal = N_VNew_OpenMP(local_NODP, num_threads,
0062     envelopeLattice.sunctx);
0063 patchToMold.duLocal = N_VNew_OpenMP(local_NODP, num_threads,
0064     envelopeLattice.sunctx);
0065 // MPI+X NVectors containing local OpenMP NVectors
0066 patchToMold.u = N_VMake_MPIPlusX(envelopeLattice.comm, patchToMold.uLocal,
0067     envelopeLattice.sunctx);
0068 patchToMold.du = N_VMake_MPIPlusX(envelopeLattice.comm, patchToMold.duLocal,
0069     envelopeLattice.sunctx);
0070 // Pointers to local vectors
0071 patchToMold.uData = N_VGetArrayPointer_MPIPlusX(patchToMold.u);
0072 patchToMold.duData = N_VGetArrayPointer_MPIPlusX(patchToMold.du);
0073 #else
0074 // MPI NVectors with local patch and global lattice size
0075 patchToMold.u =
0076     N_VNew_Parallel(envelopeLattice.comm, local_NODP,
0077         envelopeLattice.get_tot_noDP(), envelopeLattice.sunctx);
0078 patchToMold.du =
0079     N_VNew_Parallel(envelopeLattice.comm, local_NODP,
0080         envelopeLattice.get_tot_noDP(), envelopeLattice.sunctx);
0081 patchToMold.uData = NV_DATA_P(patchToMold.u);
0082 patchToMold.duData = NV_DATA_P(patchToMold.du);
0083 #endif
0084
0085 // Allocate space for auxiliary uAux so that the lattice and all possible
0086 // directions of ghost Layers fit
0087 const sunindextype s1 = patchToMold.nx, s2 = patchToMold.ny,
0088 s3 = patchToMold.nz;
0089 const sunindextype s_min = std::min(s1, std::min(s2, s3));
0090 patchToMold.uAux.resize(s1 * s2 * s3 / s_min * (s_min + 2 * gLW) * dPD);
0091 patchToMold.uAuxData = &patchToMold.uAux[0];
0092 patchToMold.envelopeLattice = &envelopeLattice;
0093 // Set patch "name" to process number -> only for debugging
0094 // patchToMold.ID=my_prc;
0095 // set flag
0096 patchToMold.statusFlags = FLatticePatchSetUp;
0097 patchToMold.generateTranslocationLookup();
0098 return 0;

```

```
00199 }
```

## 5.9.5 Field Documentation

### 5.9.5.1 buffData

```
std::array<sunrealtype *, 3> LatticePatch::buffData
```

pointer to spatial derivative data buffers

Definition at line 213 of file [LatticePatch.h](#).

Referenced by [initializeBuffers\(\)](#), [linear1DProp\(\)](#), [linear2DProp\(\)](#), [linear3DProp\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), and [nonlinear3DProp\(\)](#).

### 5.9.5.2 buffX

```
std::vector<sunrealtype> LatticePatch::buffX [private]
```

buffer to save spatial derivative values

Definition at line 174 of file [LatticePatch.h](#).

Referenced by [initializeBuffers\(\)](#).

### 5.9.5.3 buffY

```
std::vector<sunrealtype> LatticePatch::buffY [private]
```

buffer to save spatial derivative values

Definition at line 174 of file [LatticePatch.h](#).

Referenced by [initializeBuffers\(\)](#).

### 5.9.5.4 buffZ

```
std::vector<sunrealtype> LatticePatch::buffZ [private]
```

buffer to save spatial derivative values

Definition at line 174 of file [LatticePatch.h](#).

Referenced by [initializeBuffers\(\)](#).

### 5.9.5.5 du

```
N_Vector LatticePatch::du
```

Definition at line 201 of file [LatticePatch.h](#).

Referenced by [~LatticePatch\(\)](#).

### 5.9.5.6 duData

```
sunrealtyp* LatticePatch::duData
```

pointer to time-derivative data

Definition at line 205 of file [LatticePatch.h](#).

Referenced by [TimeEvolution::f\(\)](#), [linear1DProp\(\)](#), [linear2DProp\(\)](#), and [linear3DProp\(\)](#).

### 5.9.5.7 duLocal

```
N_Vector LatticePatch::duLocal
```

NVector for saving temporal derivatives of the field data.

Definition at line 201 of file [LatticePatch.h](#).

Referenced by [~LatticePatch\(\)](#).

### 5.9.5.8 dx

```
sunrealtyp LatticePatch::dx [private]
```

physical distance between lattice points in x-direction

Definition at line 157 of file [LatticePatch.h](#).

Referenced by [derive\(\)](#), and [getDelta\(\)](#).

### 5.9.5.9 dy

```
sunrealtype LatticePatch::dy [private]
```

physical distance between lattice points in y-direction

Definition at line 159 of file [LatticePatch.h](#).

Referenced by [derive\(\)](#), and [getDelta\(\)](#).

### 5.9.5.10 dz

```
sunrealtype LatticePatch::dz [private]
```

physical distance between lattice points in z-direction

Definition at line 161 of file [LatticePatch.h](#).

Referenced by [derive\(\)](#), and [getDelta\(\)](#).

### 5.9.5.11 envelopeLattice

```
const Lattice* LatticePatch::envelopeLattice [private]
```

pointer to the enveloping lattice

Definition at line 165 of file [LatticePatch.h](#).

Referenced by [derive\(\)](#), [derotate\(\)](#), [exchangeGhostCells\(\)](#), [generateTranslocationLookup\(\)](#), [initializeBuffers\(\)](#), [rotateToX\(\)](#), [rotateToY\(\)](#), and [rotateToZ\(\)](#).

### 5.9.5.12 gCLData

```
sunrealtype* LatticePatch::gCLData
```

pointer to halo data

Definition at line 210 of file [LatticePatch.h](#).

Referenced by [exchangeGhostCells\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.13 gCRData

```
sunrealtype * LatticePatch::gCRData
```

pointer to halo data

Definition at line 210 of file [LatticePatch.h](#).

Referenced by [exchangeGhostCells\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.14 ghostCellLeft

```
std::vector<sunrealtype> LatticePatch::ghostCellLeft [private]
```

buffer for passing ghost cell data

Definition at line 178 of file [LatticePatch.h](#).

Referenced by [exchangeGhostCells\(\)](#).

### 5.9.5.15 ghostCellLeftToSend

```
std::vector<sunrealtype> LatticePatch::ghostCellLeftToSend [private]
```

buffer for passing ghost cell data

Definition at line 178 of file [LatticePatch.h](#).

Referenced by [exchangeGhostCells\(\)](#).

### 5.9.5.16 ghostCellRight

```
std::vector<sunrealtype> LatticePatch::ghostCellRight [private]
```

buffer for passing ghost cell data

Definition at line 178 of file [LatticePatch.h](#).

Referenced by [exchangeGhostCells\(\)](#).

### 5.9.5.17 ghostCellRightToSend

```
std::vector<sunrealtype> LatticePatch::ghostCellRightToSend [private]
```

buffer for passing ghost cell data

Definition at line 179 of file [LatticePatch.h](#).

Referenced by [exchangeGhostCells\(\)](#).

### 5.9.5.18 ghostCells

```
std::vector<sunrealtype> LatticePatch::ghostCells [private]
```

buffer for passing ghost cell data

Definition at line 179 of file [LatticePatch.h](#).

### 5.9.5.19 ghostCellsToSend

```
std::vector<sunrealtype> LatticePatch::ghostCellsToSend [private]
```

buffer for passing ghost cell data

Definition at line 179 of file [LatticePatch.h](#).

### 5.9.5.20 ID

```
int LatticePatch::ID
```

ID of the [LatticePatch](#), corresponds to process number (for debugging)

Definition at line 197 of file [LatticePatch.h](#).

### 5.9.5.21 lgcTox

```
std::vector<sunindextype> LatticePatch::lgcTox [private]
```

ghost cell translocation lookup table

Definition at line 183 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.22 lgcToy

```
std::vector<sunindextype> LatticePatch::lgcToy [private]
```

ghost cell translocation lookup table

Definition at line 183 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.23 lgcToz

```
std::vector<sunindextype> LatticePatch::lgcToz [private]
```

ghost cell translocation lookup table

Definition at line 183 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.24 Llx

```
sunindextype LatticePatch::Llx [private]
```

inner position of lattice-patch in the lattice patchwork; x-points

Definition at line 139 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#).

### 5.9.5.25 Lly

```
sunindextype LatticePatch::Lly [private]
```

inner position of lattice-patch in the lattice patchwork; y-points

Definition at line 141 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#).

### 5.9.5.26 L<sub>z</sub>

```
sunindextype LatticePatch::Lz [private]
```

inner position of lattice-patch in the lattice patchwork; z-points

Definition at line 143 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#).

### 5.9.5.27 l<sub>x</sub>

```
sunrealtype LatticePatch::lx [private]
```

physical size of the lattice-patch in the x-dimension

Definition at line 145 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#).

### 5.9.5.28 l<sub>y</sub>

```
sunrealtype LatticePatch::ly [private]
```

physical size of the lattice-patch in the y-dimension

Definition at line 147 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#).

### 5.9.5.29 l<sub>z</sub>

```
sunrealtype LatticePatch::lz [private]
```

physical size of the lattice-patch in the z-dimension

Definition at line 149 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#).

**5.9.5.30 nx**

```
sunindextype LatticePatch::nx [private]
```

number of points in the lattice patch in the x-dimension

Definition at line 151 of file [LatticePatch.h](#).

Referenced by [discreteSize\(\)](#), [exchangeGhostCells\(\)](#), [generateTranslocationLookup\(\)](#), [initializeBuffers\(\)](#), and [LatticePatch\(\)](#).

**5.9.5.31 ny**

```
sunindextype LatticePatch::ny [private]
```

number of points in the lattice patch in the y-dimension

Definition at line 153 of file [LatticePatch.h](#).

Referenced by [discreteSize\(\)](#), [exchangeGhostCells\(\)](#), [generateTranslocationLookup\(\)](#), [initializeBuffers\(\)](#), and [LatticePatch\(\)](#).

**5.9.5.32 nz**

```
sunindextype LatticePatch::nz [private]
```

number of points in the lattice patch in the z-dimension

Definition at line 155 of file [LatticePatch.h](#).

Referenced by [discreteSize\(\)](#), [exchangeGhostCells\(\)](#), [generateTranslocationLookup\(\)](#), [initializeBuffers\(\)](#), and [LatticePatch\(\)](#).

**5.9.5.33 rgcTox**

```
std::vector<sunindextype> LatticePatch::rgcTox [private]
```

ghost cell translocation lookup table

Definition at line 183 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.34 `rgcToy`

```
std::vector<sunindextype> LatticePatch::rgcToy [private]
```

ghost cell translocation lookup table

Definition at line 183 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.35 `rgcToz`

```
std::vector<sunindextype> LatticePatch::rgcToz [private]
```

ghost cell translocation lookup table

Definition at line 183 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.36 `statusFlags`

```
unsigned int LatticePatch::statusFlags [private]
```

lattice patch status flags

Definition at line 163 of file [LatticePatch.h](#).

Referenced by [checkFlag\(\)](#), [exchangeGhostCells\(\)](#), [generateTranslocationLookup\(\)](#), [initializeBuffers\(\)](#), [LatticePatch\(\)](#), and [~LatticePatch\(\)](#).

### 5.9.5.37 `u`

```
N_Vector LatticePatch::u
```

Definition at line 199 of file [LatticePatch.h](#).

Referenced by [Simulation::advanceToTime\(\)](#), [Simulation::initializeCVODEobject\(\)](#), and [~LatticePatch\(\)](#).

### 5.9.5.38 uAux

```
std::vector<sunrealtype> LatticePatch::uAux [private]  
aid (auxilliarly) vector including ghost cells to compute the derivatives
```

Definition at line 167 of file [LatticePatch.h](#).

Referenced by [derive\(\)](#), and [derotate\(\)](#).

### 5.9.5.39 uAuxData

```
sunrealtype* LatticePatch::uAuxData
```

pointer to auxilary data vector

Definition at line 207 of file [LatticePatch.h](#).

Referenced by [rotateIntoEigen\(\)](#).

### 5.9.5.40 uData

```
sunrealtype* LatticePatch::uData
```

pointer to field data

Definition at line 203 of file [LatticePatch.h](#).

Referenced by [Simulation::addInitialConditions\(\)](#), [exchangeGhostCells\(\)](#), [TimeEvolution::f\(\)](#), [OutputManager::outUState\(\)](#), [rotateIntoEigen\(\)](#), and [Simulation::setInitialConditions\(\)](#).

### 5.9.5.41 uLocal

```
N_Vector LatticePatch::uLocal
```

NVector for saving field components  $u=(E,B)$  in lattice points.

Definition at line 199 of file [LatticePatch.h](#).

Referenced by [~LatticePatch\(\)](#).

### 5.9.5.42 uTox

```
std::vector<sunindextype> LatticePatch::uTox [private]
```

translocation lookup table

Definition at line 170 of file [LatticePatch.h](#).

Referenced by [derotate\(\)](#), [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.43 uToy

```
std::vector<sunindextype> LatticePatch::uToy [private]
```

translocation lookup table

Definition at line 170 of file [LatticePatch.h](#).

Referenced by [derotate\(\)](#), [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.44 uToz

```
std::vector<sunindextype> LatticePatch::uToz [private]
```

translocation lookup table

Definition at line 170 of file [LatticePatch.h](#).

Referenced by [derotate\(\)](#), [generateTranslocationLookup\(\)](#), and [rotateIntoEigen\(\)](#).

### 5.9.5.45 x0

```
sunrealtype LatticePatch::x0 [private]
```

origin of the patch in physical space; x-coordinate

Definition at line 133 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#), and [origin\(\)](#).

### 5.9.5.46 xTou

```
std::vector<sunindextype> LatticePatch::xTou [private]
```

translocation lookup table

Definition at line 170 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#).

### 5.9.5.47 y0

```
sunrealtype LatticePatch::y0 [private]
```

origin of the patch in physical space; y-coordinate

Definition at line 135 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#), and [origin\(\)](#).

### 5.9.5.48 yTou

```
std::vector<sunindextype> LatticePatch::yTou [private]
```

translocation lookup table

Definition at line 170 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#).

### 5.9.5.49 z0

```
sunrealtype LatticePatch::z0 [private]
```

origin of the patch in physical space; z-coordinate

Definition at line 137 of file [LatticePatch.h](#).

Referenced by [LatticePatch\(\)](#), and [origin\(\)](#).

### 5.9.5.50 zTou

```
std::vector<sunindextype> LatticePatch::zTou [private]
```

translocation lookup table

Definition at line 170 of file [LatticePatch.h](#).

Referenced by [generateTranslocationLookup\(\)](#).

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

- src/[LatticePatch.h](#)
- src/[LatticePatch.cpp](#)

## 5.10 OutputManager Class Reference

Output Manager class to generate and coordinate output writing to disk.

```
#include <src/Outputters.h>
```

### Public Member Functions

- [OutputManager \(\)](#)  
*default constructor*
- void [generateOutputFolder \(const std::string &dir\)](#)  
*function that creates folder to save simulation data*
- void [set\\_outputStyle \(const char \\_outputStyle\)](#)  
*set the output style*
- void [outUState \(const int &state, const Lattice &lattice, const LatticePatch &latticePatch\)](#)  
*function to write data to disk in specified way*
- const std::string & [getSimCode \(\) const](#)  
*simCode getter function*

### Static Private Member Functions

- static std::string [SimCodeGenerator \(\)](#)  
*function to create the Code of the Simulations*

### Private Attributes

- std::string [simCode](#)  
*variable to save the SimCode generated at execution*
- std::string [Path](#)  
*variable for the path to the output folder*
- char [outputStyle](#)  
*output style; csv or binary*

### 5.10.1 Detailed Description

Output Manager class to generate and coordinate output writing to disk.

Definition at line 21 of file [Outputters.h](#).

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 OutputManager()

```
OutputManager::OutputManager ( )
```

default constructor

Directly generate the simCode at construction.

Definition at line 12 of file [Outputters.cpp](#).

```
00012     {  
00013     simCode = SimCodeGenerator();  
00014     outputStyle = 'c';  
00015 }
```

References [outputStyle](#), [simCode](#), and [SimCodeGenerator\(\)](#).

Here is the call graph for this function:



### 5.10.3 Member Function Documentation

### 5.10.3.1 generateOutputFolder()

```
void OutputManager::generateOutputFolder (
    const std::string & dir )
```

function that creates folder to save simulation data

Generate the folder to save the data to by one process: In the given directory it creates a direcory "SimResults" and a directory with the simCode. The relevant part of the main file is written to a "config.txt" file in that directory to log the settings.

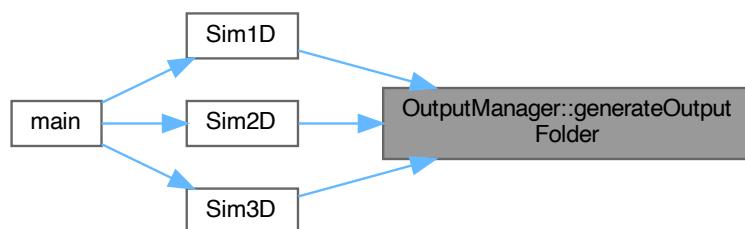
Definition at line 47 of file [Outputters.cpp](#).

```
00047
00048 // Do this only once for the first process
00049 int myPrc;
00050 MPI_Comm_rank(MPI_COMM_WORLD, &myPrc);
00051 if (myPrc == 0) {
00052     if (!fs::is_directory(dir))
00053         fs::create_directory(dir);
00054     if (!fs::is_directory(dir + "/SimResults"))
00055         fs::create_directory(dir + "/SimResults");
00056     if (!fs::is_directory(dir + "/SimResults/" + simCode))
00057         fs::create_directory(dir + "/SimResults/" + simCode);
00058 }
00059 // path variable for the output generation
00060 Path = dir + "/SimResults/" + simCode + "/";
00061
00062 // Logging configurations from main.cpp
00063 std::ifstream fin("main.cpp");
00064 std::ofstream fout(Path + "config.txt");
00065 std::string line;
00066 int begin=1000;
00067 for (int i = 1; !fin.eof(); i++) {
00068     getline(fin, line);
00069     if (line.starts_with("//----- B")) {
00070         begin=i;
00071     }
00072     if (i < begin) {
00073         continue;
00074     }
00075     fout << line << std::endl;
00076     if (line.starts_with("//----- E")) {
00077         break;
00078     }
00079 }
00080 return;
00081 }
```

References [Path](#), and [simCode](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



### 5.10.3.2 getSimCode()

```
const std::string & OutputManager::getSimCode ( ) const [inline]
```

simCode getter function

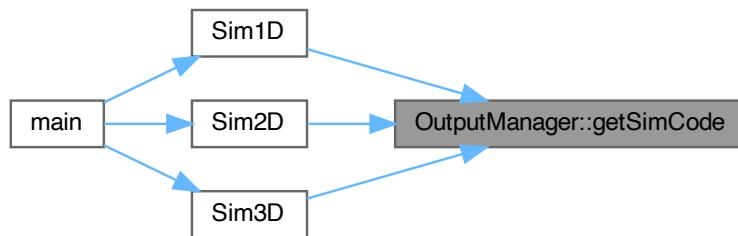
Definition at line 42 of file [Outputters.h](#).

```
00042 { return simCode; }
```

References [simCode](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



### 5.10.3.3 outUState()

```
void OutputManager::outUState (
    const int & state,
    const Lattice & lattice,
    const LatticePatch & latticePatch )
```

function to write data to disk in specified way

Write the field data either in csv format to one file per each process (patch) or in binary form to a single file. Files are stores in the simCode directory. For csv files the state (simulation step) denotes the prefix and the suffix after an underscore is given by the process/patch number. Binary files are simply named after the step number.

Definition at line 92 of file [Outputters.cpp](#).

```
00093 {
00094     switch(outputStyle){
00095         case 'c': { // one csv file per process
00096             std::ofstream ofs;
00097             ofs.open(Path + std::to_string(state) + "_"
00098                     + std::to_string(lattice.my_prc) + ".csv");
00099             // Precision of sunrealtypes in significant decimal digits; 15 for IEEE double
00100             ofs << std::setprecision(std::numeric_limits<sunrealtypes>::digits10);
00101
00102             // Walk through each lattice point
00103             const sunindextype totalNP = latticePatch.discreteSize();
00104             for (sunindextype i = 0; i < totalNP * 6; i += 6) {
00105                 // Six columns to contain the field data: Ex,Ey,Ez,Bx,By,Bz
00106                 ofs << latticePatch.uData[i + 0] << "," << latticePatch.uData[i + 1] << ","
00107             }
00108         }
00109     }
00110 }
```

```

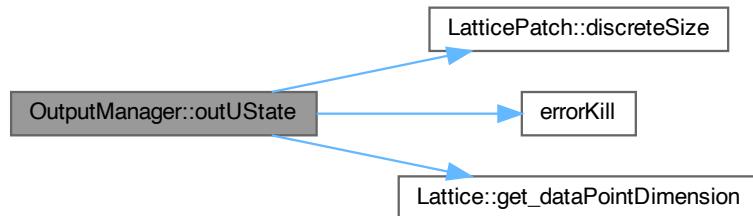
00107     << latticePatch.uData[i + 2] << ","
00108     << latticePatch.uData[i + 4] << ","
00109     << latticePatch.uData[i + 5]
00110 }
00111 ofs.close();
00112 break;
00113 }
00114
00115 case 'b': { // a single binary file
00116 // Open the output file
00117 MPI_File fh;
00118 const std::string filename = Path+std::to_string(state);
00119 MPI_File_open(lattice.comm,&filename[0],MPI_MODE_WRONLY|MPI_MODE_CREATE,
00120 MPI_INFO_NULL,&fh);
00121 // number of datapoints in the patch with process offset
00122 const sunindextype count = latticePatch.discreteSize()*
00123     lattice.get_dataPointDimension();
00124 MPI_Offset offset = lattice.my_prc*count*sizeof(MPI_SUNREALTYPE);
00125 // Go to offset in file and write data to it; maximal precision in
00126 // "native" representation
00127 MPI_File_set_view(fh,offset,MPI_SUNREALTYPE,MPI_SUNREALTYPE,"native",
00128 MPI_INFO_NULL);
00129 MPI_Request write_request;
00130 MPI_File_iwrite_all(fh,latticePatch.uData,count,MPI_SUNREALTYPE,
00131     &write_request);
00132 MPI_Wait(&write_request,MPI_STATUS_IGNORE);
00133 MPI_File_close(&fh);
00134 break;
00135 }
00136 default: {
00137     errorKill("No valid output style defined."
00138             " Choose between (c): one csv file per process,"
00139             " (b) one binary file");
00140 break;
00141 }
00142 }
00143 }

```

References [Lattice::comm](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [Lattice::get\\_dataPointDimension\(\)](#), [Lattice::my\\_prc](#), [outputStyle](#), [Path](#), and [LatticePatch::uData](#).

Referenced by [Simulation::outAllFieldData\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.10.3.4 set\_outputStyle()

```
void OutputManager::set_outputStyle (
    const char _outputStyle )
```

set the output style

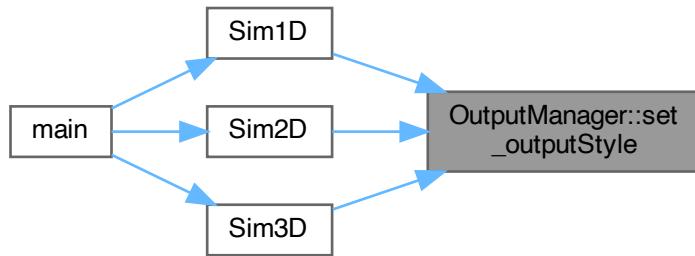
Definition at line 83 of file [Outputters.cpp](#).

```
00083
00084     outputStyle = _outputStyle;
00085 }
```

References [outputStyle](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



### 5.10.3.5 SimCodeGenerator()

```
std::string OutputManager::SimCodeGenerator ( ) [static], [private]
```

function to create the Code of the Simulations

Generate the identifier number reverse from year to minute in the format yy-mm-dd\_hh-MM-ss

Definition at line 19 of file [Outputters.cpp](#).

```
00019
00020     const chrono::time_point<chrono::system_clock> now{
00021         chrono::system_clock::now();
00022     const chrono::year_month_day ymd{chrono::floor<chrono::days>(now)};
00023     const auto tod = now - chrono::floor<chrono::days>(now);
00024     const chrono::hh_mm_ss hms{tod};
00025
00026     std::stringstream temp;
00027     temp << std::setfill('0') << std::setw(2)
00028         << static_cast<int>(ymd.year() - chrono::years(2000)) << "-"
00029         << std::setfill('0') << std::setw(2)
00030         << static_cast<unsigned>(ymd.month()) << "-"
```

```

00031     << std::setfill('0') << std::setw(2)
00032     << static_cast<unsigned>(ymd.day()) << "_"
00033     << std::setfill('0') << std::setw(2) << hms.hours().count()
00034     << "-" << std::setfill('0')
00035     << std::setw(2) << hms.minutes().count() << "-"
00036     << std::setfill('0') << std::setw(2)
00037     << hms.seconds().count();
00038     //<< "_" << hms.subseconds().count(); // subseconds render the filename
00039     // too large
00040     return temp.str();
00041 }
```

Referenced by [OutputManager\(\)](#).

Here is the caller graph for this function:



## 5.10.4 Field Documentation

### 5.10.4.1 outputStyle

char OutputManager::outputStyle [private]

output style; csv or binary

Definition at line 30 of file [Outputters.h](#).

Referenced by [OutputManager\(\)](#), [outUState\(\)](#), and [set\\_outputStyle\(\)](#).

### 5.10.4.2 Path

std::string OutputManager::Path [private]

variable for the path to the output folder

Definition at line 28 of file [Outputters.h](#).

Referenced by [generateOutputFolder\(\)](#), and [outUState\(\)](#).

### 5.10.4.3 simCode

`std::string OutputManager::simCode [private]`

variable to save the SimCode generated at execution

Definition at line 26 of file [Outputters.h](#).

Referenced by [generateOutputFolder\(\)](#), [getSimCode\(\)](#), and [OutputManager\(\)](#).

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

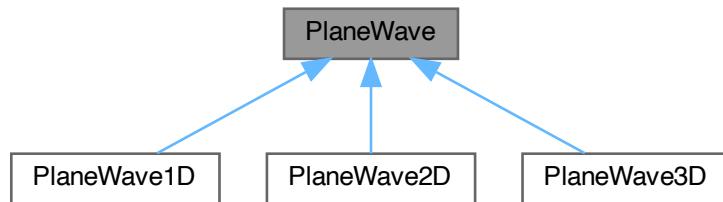
- [src/Outputters.h](#)
- [src/Outputters.cpp](#)

## 5.11 PlaneWave Class Reference

super-class for plane waves

```
#include <src/ICSetters.h>
```

Inheritance diagram for PlaneWave:



### Protected Attributes

- sunrealtype `kx`  
*wavenumber  $k_x$*
- sunrealtype `ky`  
*wavenumber  $k_y$*
- sunrealtype `kz`  
*wavenumber  $k_z$*
- sunrealtype `px`  
*polarization & amplitude in x-direction,  $p_x$*
- sunrealtype `py`  
*polarization & amplitude in y-direction,  $p_y$*
- sunrealtype `pz`  
*polarization & amplitude in z-direction,  $p_z$*
- sunrealtype `phix`  
*phase shift in x-direction,  $\phi_x$*
- sunrealtype `phiy`  
*phase shift in y-direction,  $\phi_y$*
- sunrealtype `phiz`  
*phase shift in z-direction,  $\phi_z$*

### 5.11.1 Detailed Description

super-class for plane waves

They are given in the form  $\vec{E} = \vec{E}_0 \cos(\vec{k} \cdot \vec{x} - \phi)$

Definition at line 20 of file [ICSetters.h](#).

### 5.11.2 Field Documentation

#### 5.11.2.1 kx

```
sunrealtype PlaneWave::kx [protected]
```

wavenumber  $k_x$

Definition at line 23 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

#### 5.11.2.2 ky

```
sunrealtype PlaneWave::ky [protected]
```

wavenumber  $k_y$

Definition at line 25 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

#### 5.11.2.3 kz

```
sunrealtype PlaneWave::kz [protected]
```

wavenumber  $k_z$

Definition at line 27 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

### 5.11.2.4 phix

```
sunrealtype PlaneWave::phix [protected]
```

phase shift in x-direction,  $\phi_x$

Definition at line 35 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

### 5.11.2.5 phiy

```
sunrealtype PlaneWave::phiy [protected]
```

phase shift in y-direction,  $\phi_y$

Definition at line 37 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

### 5.11.2.6 phiz

```
sunrealtype PlaneWave::phiz [protected]
```

phase shift in z-direction,  $\phi_z$

Definition at line 39 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

### 5.11.2.7 px

```
sunrealtype PlaneWave::px [protected]
```

polarization & amplitude in x-direction,  $p_x$

Definition at line 29 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

### 5.11.2.8 py

`sunrealtype PlaneWave::py [protected]`

polarization & amplitude in y-direction,  $p_y$

Definition at line 31 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

### 5.11.2.9 pz

`sunrealtype PlaneWave::pz [protected]`

polarization & amplitude in z-direction,  $p_z$

Definition at line 33 of file [ICSetters.h](#).

Referenced by [PlaneWave1D::addToSpace\(\)](#), [PlaneWave2D::addToSpace\(\)](#), [PlaneWave3D::addToSpace\(\)](#), [PlaneWave1D::PlaneWave1D\(\)](#), [PlaneWave2D::PlaneWave2D\(\)](#), and [PlaneWave3D::PlaneWave3D\(\)](#).

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

- `src/ICSetters.h`

## 5.12 planewave Struct Reference

plane wave structure

```
#include <src/SimulationFunctions.h>
```

### Data Fields

- `std::array< unrealtype, 3 > k`
- `std::array< unrealtype, 3 > p`
- `std::array< unrealtype, 3 > phi`

### 5.12.1 Detailed Description

plane wave structure

Definition at line 19 of file [SimulationFunctions.h](#).

## 5.12.2 Field Documentation

### 5.12.2.1 k

std::array<sunrealtype, 3> planewave::k

wavevector (normalized to  $1/\lambda$ )

Definition at line 20 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

### 5.12.2.2 p

std::array<sunrealtype, 3> planewave::p

amplitde & polarization vector

Definition at line 21 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

### 5.12.2.3 phi

std::array<sunrealtype, 3> planewave::phi

phase shift

Definition at line 22 of file [SimulationFunctions.h](#).

Referenced by [main\(\)](#).

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

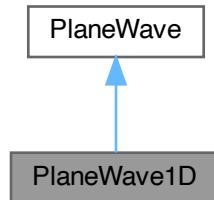
- src/[SimulationFunctions.h](#)

## 5.13 PlaneWave1D Class Reference

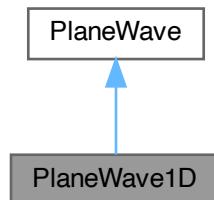
class for plane waves in 1D

```
#include <src/ICSetters.h>
```

Inheritance diagram for PlaneWave1D:



Collaboration diagram for PlaneWave1D:



### Public Member Functions

- **PlaneWave1D** (std::array< sunrealtype, 3 > k={1, 0, 0}, std::array< sunrealtype, 3 > p={0, 0, 1}, std::array< sunrealtype, 3 > phi={0, 0, 0})  
*construction with default parameters*
- void **addToSpace** (sunrealtype x, sunrealtype y, sunrealtype z, sunrealtype \*pTo6Space) const  
*function for the actual implementation in the lattice*

### Additional Inherited Members

#### 5.13.1 Detailed Description

class for plane waves in 1D

Definition at line 43 of file [ICSetters.h](#).

## 5.13.2 Constructor & Destructor Documentation

### 5.13.2.1 PlaneWave1D()

```
PlaneWave1D::PlaneWave1D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

construction with default parameters

[PlaneWave1D](#) construction with

- wavevectors  $k_x$
- $k_y$
- $k_z$  normalized to  $1/\lambda$
- amplitude (polarization) in x-direction  $p_x$
- amplitude (polarization) in y-direction  $p_y$
- amplitude (polarization) in z-direction  $p_z$
- phase shift in x-direction  $\phi_x$
- phase shift in y-direction  $\phi_y$
- phase shift in z-direction  $\phi_z$

Definition at line 11 of file [ICSetters.cpp](#).

```
00013     {
00014     kx = k[0]; /** - wavevectors \f$ k_x \f$ */
00015     ky = k[1]; /** - \f$ k_y \f$ */
00016     kz = k[2]; /** - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00017     // Amplitude bug: lower by factor 3
00018     px = p[0] / 3; /** - amplitude (polarization) in x-direction \f$ p_x \f$ */
00019     py = p[1] / 3; /** - amplitude (polarization) in y-direction \f$ p_y \f$ */
00020     pz = p[2] / 3; /** - amplitude (polarization) in z-direction \f$ p_z \f$ */
00021     phix = phi[0]; /** - phase shift in x-direction \f$ \phi_x \f$ */
00022     phiy = phi[1]; /** - phase shift in y-direction \f$ \phi_y \f$ */
00023     phiz = phi[2]; /** - phase shift in z-direction \f$ \phi_z \f$ */
00024 }
```

References [PlaneWave::kx](#), [PlaneWave::ky](#), [PlaneWave::kz](#), [PlaneWave::phix](#), [PlaneWave::phiy](#), [PlaneWave::phiz](#), [PlaneWave::px](#), [PlaneWave::py](#), and [PlaneWave::pz](#).

## 5.13.3 Member Function Documentation

### 5.13.3.1 addToSpace()

```
void PlaneWave1D::addToSpace (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space ) const
```

function for the actual implementation in the lattice

[PlaneWave1D](#) implementation in space

Definition at line 27 of file [ICSetters.cpp](#).

```
00029     {
00030     const sunrealtype wavelength =
00031         sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00032     const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00033         std::numbers::pi; /* \f$ 2\pi \cdot \vec{k} \cdot \vec{x} \f$ */
00034     // Plane wave definition
00035     const std::array<sunrealtype, 3> E{{
00036         px * cos(kScalarX - phix), /* \f$ E_x \f$ */
00037         py * cos(kScalarX - phiy), /* \f$ E_y \f$ */
00038         pz * cos(kScalarX - phiz)} }; /* \f$ E_z \f$ */
00039     // Put E-field into space
00040     pTo6Space[0] += E[0];
00041     pTo6Space[1] += E[1];
00042     pTo6Space[2] += E[2];
00043     // and B-field
00044     pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00045     pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00046     pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00047 }
```

References [PlaneWave::kx](#), [PlaneWave::ky](#), [PlaneWave::kz](#), [PlaneWave::phix](#), [PlaneWave::phiy](#), [PlaneWave::phiz](#), [PlaneWave::px](#), [PlaneWave::py](#), and [PlaneWave::pz](#).

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

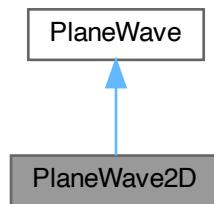
- [src/ICSetters.h](#)
- [src/ICSetters.cpp](#)

## 5.14 PlaneWave2D Class Reference

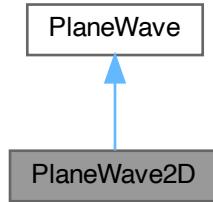
class for plane waves in 2D

```
#include <src/ICSetters.h>
```

Inheritance diagram for PlaneWave2D:



Collaboration diagram for PlaneWave2D:



## Public Member Functions

- [PlaneWave2D](#) (std::array< sunrealtype, 3 > k={1, 0, 0}, std::array< sunrealtype, 3 > p={0, 0, 1}, std::array< sunrealtype, 3 > phi={0, 0, 0})  
*construction with default parameters*
- void [addToSpace](#) (sunrealtype x, sunrealtype y, sunrealtype z, sunrealtype \*pTo6Space) const  
*function for the actual implementation in the lattice*

## Additional Inherited Members

### 5.14.1 Detailed Description

class for plane waves in 2D

Definition at line 55 of file [ICSetters.h](#).

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 PlaneWave2D()

```

PlaneWave2D::PlaneWave2D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
  
```

construction with default parameters

[PlaneWave2D](#) construction with

- wavevectors  $k_x$

- $k_y$
- $k_z$  normalized to  $1/\lambda$
- amplitude (polarization) in x-direction  $p_x$
- amplitude (polarization) in y-direction  $p_y$
- amplitude (polarization) in z-direction  $p_z$
- phase shift in x-direction  $\phi_x$
- phase shift in y-direction  $\phi_y$
- phase shift in z-direction  $\phi_z$

Definition at line 50 of file [ICSetters.cpp](#).

```

00052   {
00053     kx = k[0]; /* - wavevectors k_x */
00054     ky = k[1]; /* - k_y */
00055     kz = k[2]; /* - k_z normalized to 1/\lambda */
00056 // Amplitude bug: lower by factor 9
00057     px = p[0] / 9; /* - amplitude (polarization) in x-direction p_x */
00058     py = p[1] / 9; /* - amplitude (polarization) in y-direction p_y */
00059     pz = p[2] / 9; /* - amplitude (polarization) in z-direction p_z */
00060     phix = phi[0]; /* - phase shift in x-direction \phi_x */
00061     phiy = phi[1]; /* - phase shift in y-direction \phi_y */
00062     phiz = phi[2]; /* - phase shift in z-direction \phi_z */
00063 }
```

References [PlaneWave::kx](#), [PlaneWave::ky](#), [PlaneWave::kz](#), [PlaneWave::phix](#), [PlaneWave::phiy](#), [PlaneWave::phiz](#), [PlaneWave::px](#), [PlaneWave::py](#), and [PlaneWave::pz](#).

### 5.14.3 Member Function Documentation

#### 5.14.3.1 addToSpace()

```
void PlaneWave2D::addToSpace (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space ) const
```

function for the actual implementation in the lattice

[PlaneWave2D](#) implementation in space

Definition at line 66 of file [ICSetters.cpp](#).

```

00067   {
00068     const sunrealtype wavelength =
00069       sqrt(kx * kx + ky * ky + kz * kz); /* 1/\lambda */
00070     const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 * 
00071       std::numbers::pi; /* 2\pi / |\vec{k}| \cdot \vec{e} \cdot \vec{x} */
00072 // Plane wave definition
00073     const std::array<sunrealtype, 3> E{{
00074       px * cos(kScalarX - phix), /* E_x */
00075       py * cos(kScalarX - phiy), /* E_y */
00076       pz * cos(kScalarX - phiz)} }; /* E_z */
00077 // Put E-field into space
00078     pTo6Space[0] += E[0];
00079     pTo6Space[1] += E[1];
00080     pTo6Space[2] += E[2];
00081 // and B-field
00082     pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
```

```
00083     pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00084     pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00085 }
```

References [PlaneWave::kx](#), [PlaneWave::ky](#), [PlaneWave::kz](#), [PlaneWave::phix](#), [PlaneWave::phiy](#), [PlaneWave::phiz](#), [PlaneWave::px](#), [PlaneWave::py](#), and [PlaneWave::pz](#).

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

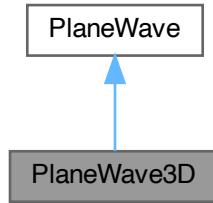
- [src/ICSetters.h](#)
- [src/ICSetters.cpp](#)

## 5.15 PlaneWave3D Class Reference

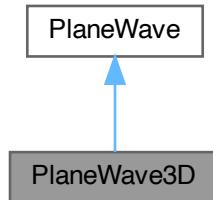
class for plane waves in 3D

```
#include <src/ICSetters.h>
```

Inheritance diagram for PlaneWave3D:



Collaboration diagram for PlaneWave3D:



## Public Member Functions

- `PlaneWave3D` (`std::array< sunrealtype, 3 > k={1, 0, 0}, std::array< sunrealtype, 3 > p={0, 0, 1}, std::array< sunrealtype, 3 > phi={0, 0, 0})`  
*construction with default parameters*
- void `addToSpace` (`sunrealtype x, sunrealtype y, sunrealtype z, sunrealtype *pTo6Space)` const  
*function for the actual implementation in space*

## Additional Inherited Members

### 5.15.1 Detailed Description

class for plane waves in 3D

Definition at line 67 of file [ICSetters.h](#).

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 `PlaneWave3D()`

```
PlaneWave3D::PlaneWave3D (
    std::array< sunrealtype, 3 > k = {1, 0, 0},
    std::array< sunrealtype, 3 > p = {0, 0, 1},
    std::array< sunrealtype, 3 > phi = {0, 0, 0} )
```

construction with default parameters

`PlaneWave3D` construction with

- wavevectors  $k_x$
- $k_y$
- $k_z$  normalized to  $1/\lambda$
- amplitude (polarization) in x-direction  $p_x$
- amplitude (polarization) in y-direction  $p_y$
- amplitude (polarization) in z-direction  $p_z$
- phase shift in x-direction  $\phi_x$
- phase shift in y-direction  $\phi_y$
- phase shift in z-direction  $\phi_z$

Definition at line 88 of file [ICSetters.cpp](#).

```
00090      {
00091     kx = k[0];    /** - wavevectors \f$ k_x \f$ */
00092     ky = k[1];    /** - \f$ k_y \f$ */
00093     kz = k[2];    /** - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00094     px = p[0];    /** - amplitude (polarization) in x-direction \f$ p_x \f$ */
00095     py = p[1];    /** - amplitude (polarization) in y-direction \f$ p_y \f$ */
00096     pz = p[2];    /** - amplitude (polarization) in z-direction \f$ p_z \f$ */
00097     phix = phi[0]; /** - phase shift in x-direction \f$ \phi_{\text{x}} \f$ */
00098     phiy = phi[1]; /** - phase shift in y-direction \f$ \phi_{\text{y}} \f$ */
00099     phiz = phi[2]; /** - phase shift in z-direction \f$ \phi_{\text{z}} \f$ */
00100 }
```

References `PlaneWave::kx`, `PlaneWave::ky`, `PlaneWave::kz`, `PlaneWave::phix`, `PlaneWave::phiy`, `PlaneWave::phiz`, `PlaneWave::px`, `PlaneWave::py`, and `PlaneWave::pz`.

### 5.15.3 Member Function Documentation

#### 5.15.3.1 addToSpace()

```
void PlaneWave3D::addToSpace (
    sunrealtype x,
    sunrealtype y,
    sunrealtype z,
    sunrealtype * pTo6Space ) const
```

function for the actual implementation in space

[PlaneWave3D](#) implementation in space

Definition at line 103 of file [ICSetters.cpp](#).

```
00104
00105     const sunrealtype wavelength =
00106         sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00107     const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00108         std::numbers::pi; /* \f$ 2\pi \vec{k} \cdot \vec{x} \f$ */
00109 // Plane wave definition
00110     const std::array<sunrealtype, 3> E{ /* E-field vector \f$ \vec{E} \f$ */
00111         px * cos(kScalarX - phix), /* \f$ E_x \f$ */
00112         py * cos(kScalarX - phiy), /* \f$ E_y \f$ */
00113         pz * cos(kScalarX - phiz)}; /* \f$ E_z \f$ */
00114 // Put E-field into space
00115     pTo6Space[0] += E[0];
00116     pTo6Space[1] += E[1];
00117     pTo6Space[2] += E[2];
00118 // and B-field
00119     pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00120     pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00121     pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00122 }
```

References [PlaneWave::kx](#), [PlaneWave::ky](#), [PlaneWave::kz](#), [PlaneWave::phix](#), [PlaneWave::phiy](#), [PlaneWave::phiz](#), [PlaneWave::px](#), [PlaneWave::py](#), and [PlaneWave::pz](#).

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

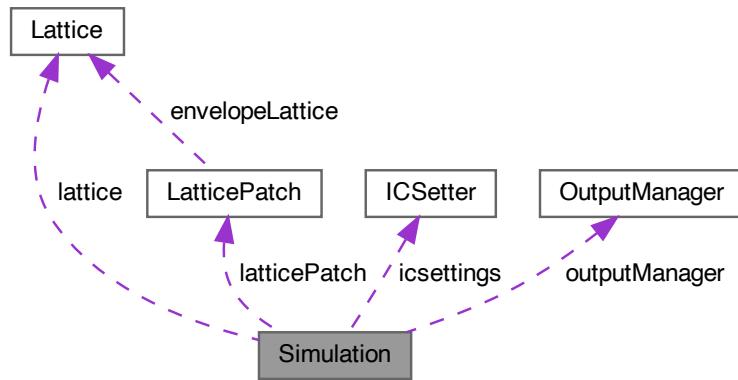
- [src/ICSetters.h](#)
- [src/ICSetters.cpp](#)

## 5.16 Simulation Class Reference

[Simulation](#) class to instantiate the whole walkthrough of a [Simulation](#).

```
#include <src/SimulationClass.h>
```

Collaboration diagram for Simulation:



## Public Member Functions

- **Simulation** (const int Nx, const int Ny, const int Nz, const int StencilOrder, const bool periodicity)
   
*constructor function for the creation of the cartesian communicator*
- **~Simulation ()**
  
*destructor function freeing CVode memory and Sundials context*
- **MPI\_Comm \* get\_cart\_comm ()**
  
*reference to the cartesian communicator of the lattice (for debugging)*
- **void setDiscreteDimensionsOfLattice (const sunindextype \_tot\_nx, const sunindextype \_tot\_ny, const sunindextype \_tot\_nz)**
  
*function to set discrete dimensions of the lattice*
- **void setPhysicalDimensionsOfLattice (const sunrealtype lx, const sunrealtype ly, const sunrealtype lz)**
  
*function to set physical dimensions of the lattice*
- **void initializePatchwork (const int nx, const int ny, const int nz)**
  
*function to initialize the Patchwork*
- **void initializeCVODEobject (const sunrealtype reltol, const sunrealtype abstol)**
  
*function to initialize the CVODE object with all requirements*
- **void start ()**
  
*function to start the simulation for time iteration*
- **void setInitialConditions ()**
  
*functions to set the initial field configuration onto the lattice*
- **void addInitialConditions (const sunindextype xm, const sunindextype ym, const sunindextype zm=0)**
  
*functions to add initial periodic field configurations*
- **void addPeriodicICLayerInX ()**
  
*function to add a periodic IC layer in one dimension*
- **void addPeriodicICLayerInXY ()**
  
*function to add periodic IC layers in two dimensions*
- **void advanceToTime (const sunrealtype &tEnd)**
  
*function to advance solution in time with CVODE*
- **void outAllFieldData (const int &state)**
  
*function to write field data to disk*

- void `checkFlag` (unsigned int flag) const  
*function to check if flag has been set*
- void `checkNoFlag` (unsigned int flag) const  
*function to check if flag has not been set*

## Data Fields

- `ICSetter icsettings`  
*IC Setter object.*
- `OutputManager outputManager`  
*Output Manager object.*
- `void * cvode_mem`  
*pointer to CVode memory object*
- `SUNNonlinearSolver NLS`  
*nonlinear solver object*

## Private Attributes

- `Lattice lattice`  
*Lattice object.*
- `LatticePatch latticePatch`  
*LatticePatch object.*
- `sunrealtype t`  
*current time of the simulation*
- `unsigned int statusFlags`  
*simulation status flags*

### 5.16.1 Detailed Description

`Simulation` class to instantiate the whole walkthrough of a `Simulation`.

Definition at line 30 of file `SimulationClass.h`.

### 5.16.2 Constructor & Destructor Documentation

### 5.16.2.1 Simulation()

```
Simulation::Simulation (
    const int Nx,
    const int Ny,
    const int Nz,
    const int StencilOrder,
    const bool periodicity )
```

constructor function for the creation of the cartesian communicator

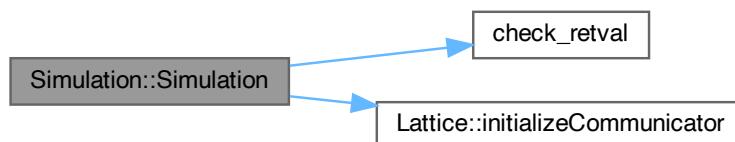
Along with the simulation object, create the cartesian communicator and SUNContext object

Definition at line 14 of file [SimulationClass.cpp](#).

```
00015 :
00016     lattice(StencilOrder) {
00017         statusFlags = 0;
00018         t = 0;
00019         // Initialize the cartesian communicator
00020         lattice.initializeCommunicator(Nx, Ny, Nz, periodicity);
00021
00022         // Create the SUNContext object associated with the thread of execution
00023         int retval = 0;
00024         retval = SUNContext_Create(&lattice.comm, &lattice.sunctx);
00025         if (check_retval(&retval, "SUNContext_Create", 1, lattice.my_prc))
00026             MPI_Abort(lattice.comm, 1);
00027 }
```

References [check\\_retval\(\)](#), [Lattice::comm](#), [Lattice::initializeCommunicator\(\)](#), [lattice](#), [Lattice::my\\_prc](#), [statusFlags](#), [Lattice::sunctx](#), and [t](#).

Here is the call graph for this function:



### 5.16.2.2 ~Simulation()

```
Simulation::~Simulation ( )
```

destructor function freeing CVode memory and Sundials context

Free the CVode solver memory and Sundials context object with the finish of the simulation

Definition at line 31 of file [SimulationClass.cpp](#).

```
00031     {
00032         // Free solver memory
00033         if (statusFlags & CvodeObjectSetUp) {
00034             CVodeFree(&cvode_mem);
00035             SUNNonlinSolFree(NLS);
00036             SUNContext_Free(&lattice.sunctx);
00037         }
00038     }
```

References [cvode\\_mem](#), [CvodeObjectSetUp](#), [lattice](#), [NLS](#), [statusFlags](#), and [Lattice::sunctx](#).

### 5.16.3 Member Function Documentation

#### 5.16.3.1 addInitialConditions()

```
void Simulation::addInitialConditions (
    const sunindextype xm,
    const sunindextype ym,
    const sunindextype zm = 0 )
```

functions to add initial periodic field configurations

Use parameters to add periodic IC layers.

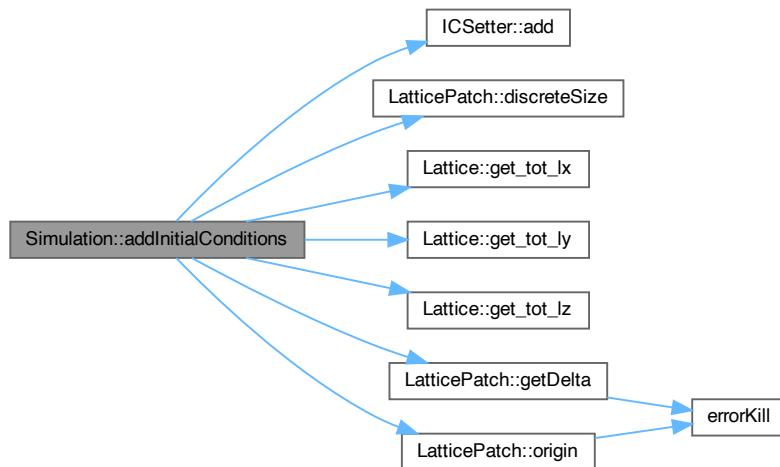
Definition at line 167 of file [SimulationClass.cpp](#).

```
00169     {
00170     const sunrealtype dx = latticePatch.getDelta(1);
00171     const sunrealtype dy = latticePatch.getDelta(2);
00172     const sunrealtype dz = latticePatch.getDelta(3);
00173     const sunindextype nx = latticePatch.discreteSize(1);
00174     const sunindextype ny = latticePatch.discreteSize(2);
00175     const sunindextype totalNP = latticePatch.discreteSize();
00176     // Correct for demanded displacement, rest as for setInitialConditions
00177     const sunrealtype x0 = latticePatch.origin(1) + xm*lattice.get_tot_lx();
00178     const sunrealtype y0 = latticePatch.origin(2) + ym*lattice.get_tot_ly();
00179     const sunrealtype z0 = latticePatch.origin(3) + zm*lattice.get_tot_lz();
00180     sunindextype px = 0, py = 0, pz = 0;
00181     for (sunindextype i = 0; i < totalNP * 6; i += 6) {
00182         px = (i / 6) % nx;
00183         py = ((i / 6) / nx) % ny;
00184         pz = ((i / 6) / nx) / ny;
00185         icsettings.add(static_cast<sunrealtype>(px) * dx + x0,
00186                         static_cast<sunrealtype>(py) * dy + y0,
00187                         static_cast<sunrealtype>(pz) * dz + z0, &latticePatch.uData[i]);
00188     }
00189     return;
00190 }
```

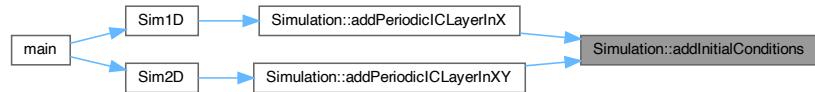
References [ICSetter::add\(\)](#), [LatticePatch::discreteSize\(\)](#), [Lattice::get\\_tot\\_lx\(\)](#), [Lattice::get\\_tot\\_ly\(\)](#), [Lattice::get\\_tot\\_lz\(\)](#), [LatticePatch::getDelta\(\)](#), [icsettings](#), [lattice](#), [latticePatch](#), [LatticePatch::origin\(\)](#), and [LatticePatch::uData](#).

Referenced by [addPeriodicICLayerInX\(\)](#), and [addPeriodicICLayerInXY\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.2 addPeriodicICLayerInX()

```
void Simulation::addPeriodicICLayerInX ( )
```

function to add a periodic IC layer in one dimension

Add initial conditions in one dimension.

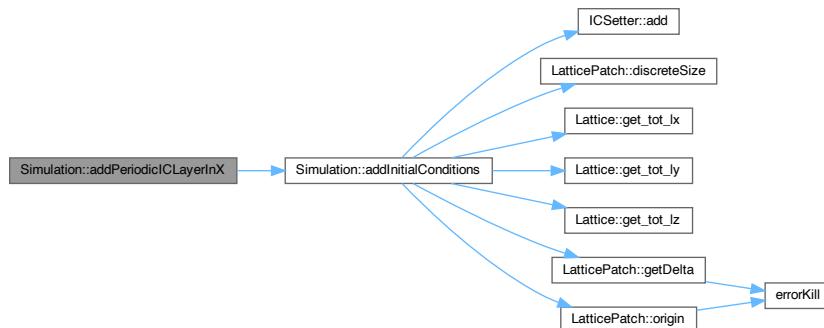
Definition at line 193 of file [SimulationClass.cpp](#).

```
00193     {
00194     addInitialConditions(-1, 0, 0);
00195     addInitialConditions(1, 0, 0);
00196     return;
00197 }
```

References [addInitialConditions\(\)](#).

Referenced by [Sim1D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.3 addPeriodicICLayerInXY()

```
void Simulation::addPeriodicICLayerInXY ( )
```

function to add periodic IC layers in two dimensions

Add initial conditions in two dimensions.

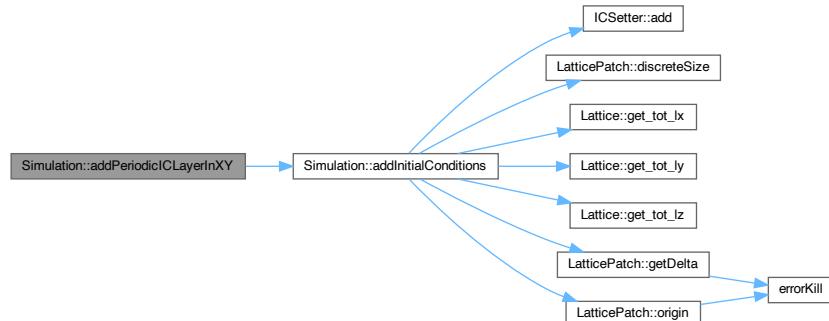
Definition at line 200 of file [SimulationClass.cpp](#).

```
00200     {
00201     addInitialConditions(-1, -1, 0);
00202     addInitialConditions(-1, 0, 0);
00203     addInitialConditions(-1, 1, 0);
00204     addInitialConditions(0, 1, 0);
00205     addInitialConditions(0, -1, 0);
00206     addInitialConditions(1, -1, 0);
00207     addInitialConditions(1, 0, 0);
00208     addInitialConditions(1, 1, 0);
00209     return;
00210 }
```

References [addInitialConditions\(\)](#).

Referenced by [Sim2D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.16.3.4 advanceToTime()

```
void Simulation::advanceToTime (
    const sunrealtype & tEnd )
```

function to advance solution in time with CVODE

Advance the solution in time -> integrate the ODE over an interval t.

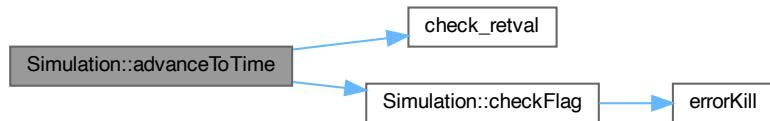
Definition at line 213 of file [SimulationClass.cpp](#).

```
00213
00214     checkFlag(SimulationStarted);
00215     int retval = 0;
00216     retval = CVode(cvode_mem, tEnd, latticePatch.u, &t,
00217                     CV_NORMAL); // CV_NORMAL: internal steps to reach tEnd, then
00218                     // interpolate to return latticePatch.u, return time
00219                     // reached by the solver as t
00220     if (check_retval(&retval, "CVode", 1, lattice.my_prc))
00221         MPI_Abort(lattice.comm, 1);
00222 }
```

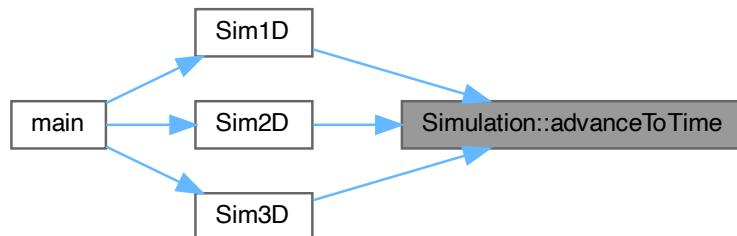
References [check\\_retval\(\)](#), [checkFlag\(\)](#), [Lattice::comm](#), [cvode\\_mem](#), [lattice](#), [latticePatch](#), [Lattice::my\\_prc](#), [SimulationStarted](#), [t](#), and [LatticePatch::u](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.5 checkFlag()

```
void Simulation::checkFlag (
    unsigned int flag ) const
```

function to check if flag has been set

Check presence of configuration flags.

Definition at line 231 of file [SimulationClass.cpp](#).

```
00231
00232     if (!(statusFlags & flag)) {
00233         std::string errorMessage;
00234         switch (flag) {
00235             case LatticeDiscreteSetUp:
00236                 errorMessage = "The discrete size of the Simulation has not been set up";
00237                 break;
00238             case LatticePhysicalSetUp:
00239                 errorMessage = "The physical size of the Simulation has not been set up";
00240                 break;
00241             case LatticePatchworkSetUp:
00242                 errorMessage = "The patchwork for the Simulation has not been set up";
00243                 break;
00244             case CvodeObjectsetUp:
00245                 errorMessage = "The CVODE object has not been initialized";
00246                 break;
00247             case SimulationStarted:
00248                 errorMessage = "The Simulation has not been started";
00249                 break;
00250             default:
00251                 errorMessage = "Uppss, you've made a non-standard error, sadly I can't "
00252                             "help you there";
00253                 break;
00254         }
00255         errorKill(errorMessage);
00256     }
00257     return;
00258 }
```

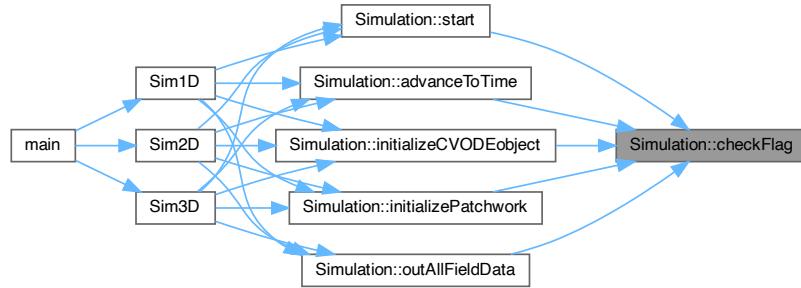
References [CvodeObjectsetUp](#), [errorKill\(\)](#), [LatticeDiscreteSetUp](#), [LatticePatchworkSetUp](#), [LatticePhysicalSetUp](#), [SimulationStarted](#), and [statusFlags](#).

Referenced by [advanceToTime\(\)](#), [initializeCVODEobject\(\)](#), [initializePatchwork\(\)](#), [outAllFieldData\(\)](#), and [start\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.6 `checkNoFlag()`

```
void Simulation::checkNoFlag (
    unsigned int flag ) const
```

function to check if flag has not been set

Check absence of configuration flags.

Definition at line 261 of file `SimulationClass.cpp`.

```
00261
00262     if ((statusFlags & flag)) {
00263         std::string errorMessage;
00264         switch (flag) {
00265             case LatticeDiscreteSetUp:
00266                 errorMessage =
00267                     "The discrete size of the Simulation has already been set up";
00268                 break;
00269             case LatticePhysicalSetUp:
00270                 errorMessage =
00271                     "The physical size of the Simulation has already been set up";
00272                 break;
00273             case LatticePatchworkSetUp:
00274                 errorMessage = "The patchwork for the Simulation has already been set up";
00275                 break;
00276             case CvodeObjectsetUp:
00277                 errorMessage = "The CVODE object has already been initialized";
00278                 break;
00279             case SimulationStarted:
00280                 errorMessage = "The simulation has already started, some changes are no "
00281                             "longer possible";
00282                 break;
00283             default:
00284                 errorMessage = "Uppss, you've made a non-standard error, sadly I can't "
00285                             "help you there";
00286                 break;
00287         }
00288         errorKill(errorMessage);
00289     }
00290     return;
00291 }
```

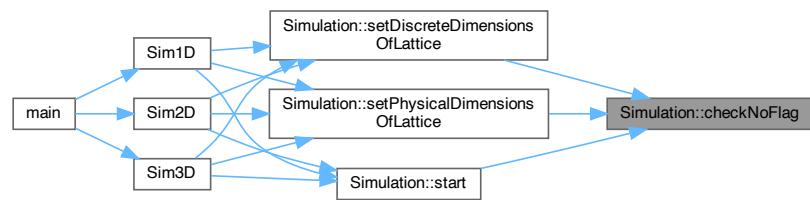
References `CvodeObjectsetUp`, `errorKill()`, `LatticeDiscreteSetUp`, `LatticePatchworkSetUp`, `LatticePhysicalSetUp`, `SimulationStarted`, and `statusFlags`.

Referenced by `setDiscreteDimensionsOfLattice()`, `setPhysicalDimensionsOfLattice()`, and `start()`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.7 get\_cart\_comm()

```
MPI_Comm * Simulation::get_cart_comm ( ) [inline]
```

reference to the cartesian communicator of the lattice (for debugging)

Definition at line 56 of file [SimulationClass.h](#).

```
00056 { return &lattice.comm; }
```

References [Lattice::comm](#), and [lattice](#).

### 5.16.3.8 initializeCVODEobject()

```
void Simulation::initializeCVODEobject (
    const sunrealtype reltol,
    const sunrealtype abstol )
```

function to initialize the CVODE object with all requirements

Configure CVODE.

Definition at line 70 of file [SimulationClass.cpp](#).

```
00071 {
00072     checkFlag(SimulationStarted);
```

```

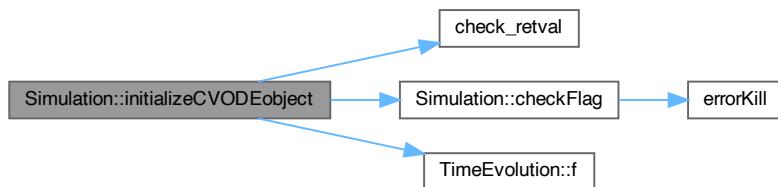
00073
00074 // CVode settings return value
00075 int retval = 0;
00076
00077 // Create CVODE object -- returns a pointer to the cvode memory structure
00078 // with Adams method (Adams-Moulton formula) solver chosen for non-stiff ODE
00079 cvode_mem = CVodeCreate(CV_ADAMS, lattice.sunctx);
00080
00081 // Specify user data and attach it to the main cvode memory block
00082 retval = CVodeSetUserData(
00083     cvode_mem,
00084     &latticePatch); // patch contains the user data as used in CVRhsFn
00085 if (check_retval(&retval, "CVodeSetUserData", 1, lattice.my_prc))
00086     MPI_Abort(lattice.comm, 1);
00087
00088 // Initialize CVODE solver
00089 retval = CVodeInit(cvode_mem, TimeEvolution::f, 0,
00090                     latticePatch.u); // allocate memory, CVRhsFn f, t_i=0, u
00091                     // contains the initial values
00092 if (check_retval(&retval, "CVodeInit", 1, lattice.my_prc))
00093     MPI_Abort(lattice.comm, 1);
00094
00095 // Create fixed point nonlinear solver object (suitable for non-stiff ODE) and
00096 // attach it to CVode
00097 NLS = SUNNonlinSol_FixedPoint(latticePatch.u, 0, lattice.sunctx);
00098 retval = CVodeSetNonlinearSolver(cvode_mem, NLS);
00099 if (check_retval(&retval, "CVodeSetNonlinearSolver", 1, lattice.my_prc))
00100     MPI_Abort(lattice.comm, 1);
00101
00102 // Anderson damping factor
00103 retval = SUNNonlinSolSetDamping_FixedPoint(NLS, 1);
00104 if (check_retval(&retval, "SUNNonlinSolSetDamping_FixedPoint", 1,
00105                 lattice.my_prc)) MPI_Abort(lattice.comm, 1);
00106
00107 // Specify integration tolerances -- a scalar relative tolerance and scalar
00108 // absolute tolerance
00109 retval = CVodeSStolerances(cvode_mem, reltol, abstol);
00110 if (check_retval(&retval, "CVodeSStolerances", 1, lattice.my_prc))
00111     MPI_Abort(lattice.comm, 1);
00112
00113 // Specify the maximum number of steps to be taken by the solver in its
00114 // attempt to reach the next tout
00115 retval = CVodeSetMaxNumSteps(cvode_mem, 10000);
00116 if (check_retval(&retval, "CVodeSetMaxNumSteps", 1, lattice.my_prc))
00117     MPI_Abort(lattice.comm, 1);
00118
00119 // maximum number of warnings for too small h
00120 retval = CVodeSetMaxHnilWarns(cvode_mem, 3);
00121 if (check_retval(&retval, "CVodeSetMaxHnilWarns", 1, lattice.my_prc))
00122     MPI_Abort(lattice.comm, 1);
00123
00124 statusFlags |= CvodeObjectSetUp;
00125 }

```

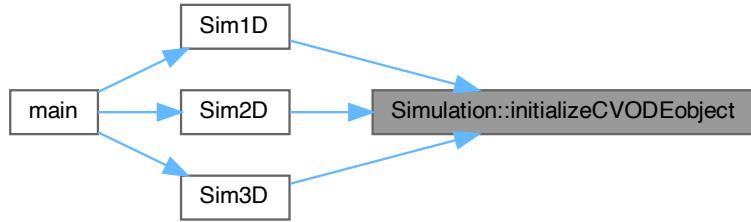
References `check_retval()`, `checkFlag()`, `Lattice::comm`, `cvode_mem`, `CvodeObjectSetUp`, `TimeEvolution::f()`, `lattice`, `latticePatch`, `Lattice::my_prc`, `NLS`, `SimulationStarted`, `statusFlags`, `Lattice::sunctx`, and `LatticePatch::u`.

Referenced by `Sim1D()`, `Sim2D()`, and `Sim3D()`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.9 initializePatchwork()

```
void Simulation::initializePatchwork (
    const int nx,
    const int ny,
    const int nz )
```

function to initialize the Patchwork

Check that the lattice dimensions are set up and generate the patchwork.

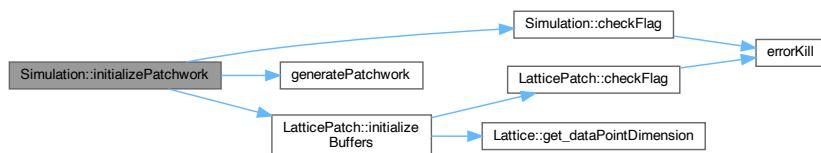
Definition at line 57 of file [SimulationClass.cpp](#).

```
00058     {
00059     checkFlag(LatticeDiscreteSetUp);
00060     checkFlag(LatticePhysicalSetUp);
00061
00062     // Generate the patchwork
00063     generatePatchwork(lattice, latticePatch, nx, ny, nz);
00064     latticePatch.initializeBuffers();
00065
00066     statusFlags |= LatticePatchworkSetUp;
00067 }
```

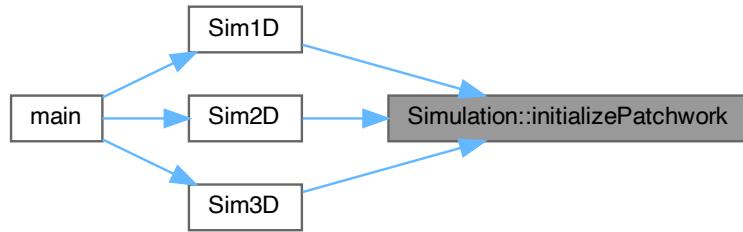
References [checkFlag\(\)](#), [generatePatchwork\(\)](#), [LatticePatch::initializeBuffers\(\)](#), [lattice](#), [LatticeDiscreteSetUp](#), [latticePatch](#), [LatticePatchworkSetUp](#), [LatticePhysicalSetUp](#), and [statusFlags](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.10 outAllFieldData()

```
void Simulation::outAllFieldData (
    const int & state )
```

function to write field data to disk

Write specified simulation steps to disk.

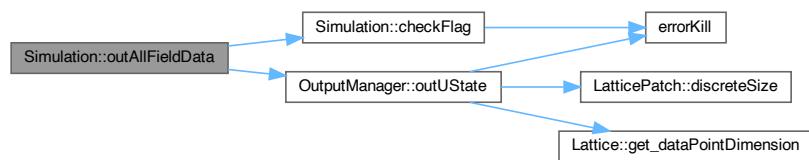
Definition at line 225 of file [SimulationClass.cpp](#).

```
00225
00226     checkFlag(SimulationStarted);
00227     outputManager.outUState(state, lattice, latticePatch);
00228 }
```

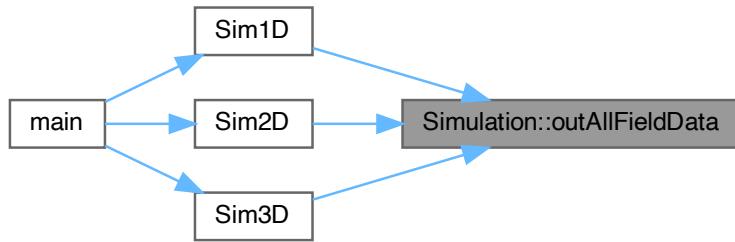
References [checkFlag\(\)](#), [lattice](#), [latticePatch](#), [outputManager](#), [OutputManager::outUState\(\)](#), and [SimulationStarted](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.11 setDiscreteDimensionsOfLattice()

```
void Simulation::setDiscreteDimensionsOfLattice (
    const sunindextype _tot_nx,
    const sunindextype _tot_ny,
    const sunindextype _tot_nz )
```

function to set discrete dimensions of the lattice

Set the discrete dimensions, the number of points per dimension.

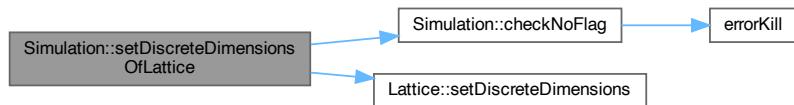
Definition at line 41 of file [SimulationClass.cpp](#).

```
00042
00043     checkNoFlag(LatticePatchworkSetUp);
00044     lattice.setDiscreteDimensions(nx, ny, nz);
00045     statusFlags |= LatticeDiscreteSetUp;
00046 }
```

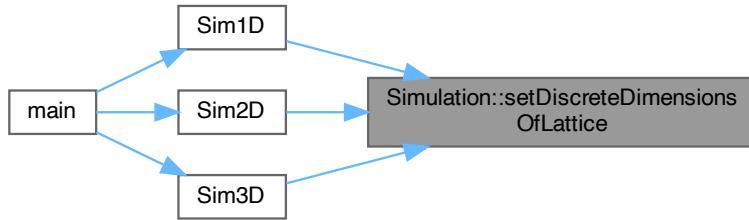
References [checkNoFlag\(\)](#), [lattice](#), [LatticeDiscreteSetUp](#), [LatticePatchworkSetUp](#), [Lattice::setDiscreteDimensions\(\)](#), and [statusFlags](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.12 setInitialConditions()

```
void Simulation::setInitialConditions ( )
```

functions to set the initial field configuration onto the lattice

Set initial conditions: Fill the lattice points with the initial field values

Definition at line 140 of file [SimulationClass.cpp](#).

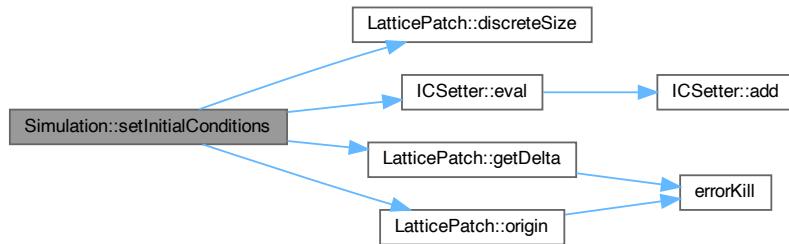
```

00140 {
00141     const sunrealtyp dx = latticePatch.getDelta(1);
00142     const sunrealtyp dy = latticePatch.getDelta(2);
00143     const sunrealtyp dz = latticePatch.getDelta(3);
00144     const sunindextyp nx = latticePatch.discreteSize(1);
00145     const sunindextyp ny = latticePatch.discreteSize(2);
00146     const sunindextyp totalNP = latticePatch.discreteSize();
00147     const sunrealtyp x0 = latticePatch.origin(1);
00148     const sunrealtyp y0 = latticePatch.origin(2);
00149     const sunrealtyp z0 = latticePatch.origin(3);
00150     sunindextyp px = 0, py = 0, pz = 0;
00151     #pragma omp parallel for default(none) \
00152     shared(nx, ny, totalNP, dx, dy, dz, x0, y0, z0) \
00153     firstprivate(px, py, pz) schedule(static)
00154     for (sunindextyp i = 0; i < totalNP * 6; i += 6) {
00155         px = (i / 6) % nx;
00156         py = ((i / 6) / nx) % ny;
00157         pz = ((i / 6) / nx) / ny;
00158         // Call the 'eval' function to fill the lattice points with the field data
00159         icsettings.eval(static_cast<sunrealtyp>(px) * dx + x0,
00160                         static_cast<sunrealtyp>(py) * dy + y0,
00161                         static_cast<sunrealtyp>(pz) * dz + z0, &latticePatch.uData[i]);
00162     }
00163     return;
00164 }
```

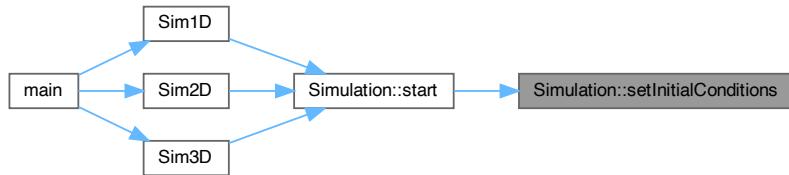
References [LatticePatch::discreteSize\(\)](#), [ICSetter::eval\(\)](#), [LatticePatch::getDelta\(\)](#), [icsettings](#), [latticePatch](#), [LatticePatch::origin\(\)](#), and [LatticePatch::uData](#).

Referenced by [start\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.16.3.13 `setPhysicalDimensionsOfLattice()`

```

void Simulation::setPhysicalDimensionsOfLattice (
    const sunrealtypelx,
    const sunrealtypely,
    const sunrealtypelz )
  
```

function to set physical dimensions of the lattice

Set the physical dimensions with lengths in micro meters.

Definition at line 49 of file [SimulationClass.cpp](#).

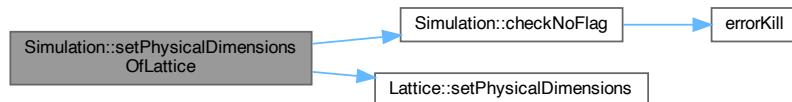
```

00050
00051     checkNoFlag(LatticePatchworkSetUp);
00052     lattice.setPhysicalDimensions(lx, ly, lz);
00053     statusFlags |= LatticePhysicalSetUp;
00054 }
  
```

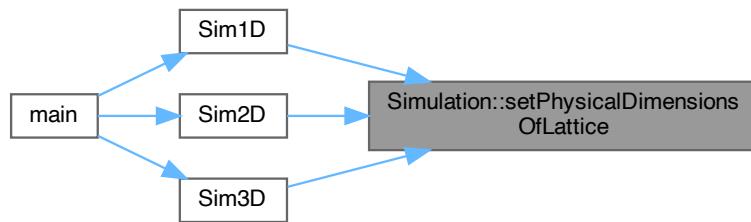
References [checkNoFlag\(\)](#), [lattice](#), [LatticePatchworkSetUp](#), [LatticePhysicalSetUp](#), [Lattice::setPhysicalDimensions\(\)](#), and [statusFlags](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.16.3.14 start()

```
void Simulation::start ( )
```

function to start the simulation for time iteration

Check if the lattice patchwork is set up and set the initial conditions.

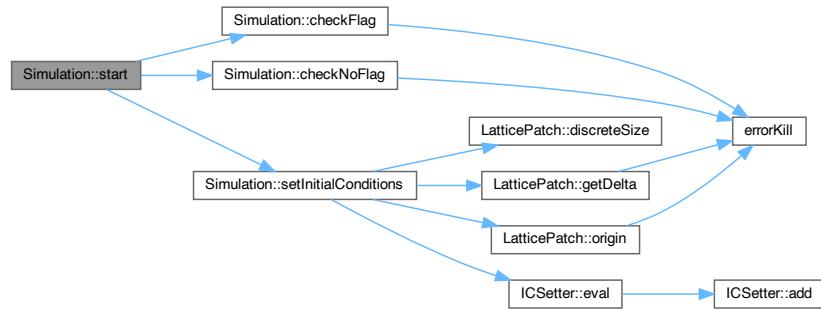
Definition at line 128 of file [SimulationClass.cpp](#).

```
00128     {
00129     checkFlag(LatticeDiscreteSetUp);
00130     checkFlag(LatticePhysicalSetUp);
00131     checkFlag(LatticePatchworkSetUp);
00132     checkNoFlag(SimulationStarted);
00133     checkNoFlag(CvodeObjectSetUp);
00134     setInitialConditions();
00135     statusFlags |= SimulationStarted;
00136 }
```

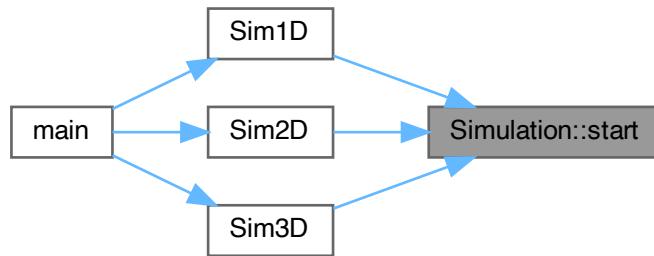
References [checkFlag\(\)](#), [checkNoFlag\(\)](#), [CvodeObjectSetUp](#), [LatticeDiscreteSetUp](#), [LatticePatchworkSetUp](#), [LatticePhysicalSetUp](#), [setInitialConditions\(\)](#), [SimulationStarted](#), and [statusFlags](#).

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.16.4 Field Documentation

### 5.16.4.1 cvode\_mem

```
void* Simulation::cvode_mem
```

pointer to CVode memory object

Definition at line 47 of file [SimulationClass.h](#).

Referenced by [advanceToTime\(\)](#), [initializeCVODEObject\(\)](#), and [~Simulation\(\)](#).

#### 5.16.4.2 icsettings

`ICSetter` `Simulation::icsettings`

IC Setter object.

Definition at line 43 of file [SimulationClass.h](#).

Referenced by [addInitialConditions\(\)](#), [setInitialConditions\(\)](#), [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

#### 5.16.4.3 lattice

`Lattice` `Simulation::lattice` [private]

Lattice object.

Definition at line 33 of file [SimulationClass.h](#).

Referenced by [addInitialConditions\(\)](#), [advanceToTime\(\)](#), [get\\_cart\\_comm\(\)](#), [initializeCVODEobject\(\)](#), [initializePatchwork\(\)](#), [outAllFieldData\(\)](#), [setDiscreteDimensionsOfLattice\(\)](#), [setPhysicalDimensionsOfLattice\(\)](#), [Simulation\(\)](#), and [~Simulation\(\)](#).

#### 5.16.4.4 latticePatch

`LatticePatch` `Simulation::latticePatch` [private]

LatticePatch object.

Definition at line 35 of file [SimulationClass.h](#).

Referenced by [addInitialConditions\(\)](#), [advanceToTime\(\)](#), [initializeCVODEobject\(\)](#), [initializePatchwork\(\)](#), [outAllFieldData\(\)](#), and [setInitialConditions\(\)](#).

#### 5.16.4.5 NLS

`SUNNonlinearSolver` `Simulation::NLS`

nonlinear solver object

Definition at line 49 of file [SimulationClass.h](#).

Referenced by [initializeCVODEobject\(\)](#), and [~Simulation\(\)](#).

#### 5.16.4.6 outputManager

`OutputManager Simulation::outputManager`

Output Manager object.

Definition at line 45 of file [SimulationClass.h](#).

Referenced by [outAllFieldData\(\)](#), [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

#### 5.16.4.7 statusFlags

`unsigned int Simulation::statusFlags [private]`

simulation status flags

Definition at line 39 of file [SimulationClass.h](#).

Referenced by [checkFlag\(\)](#), [checkNoFlag\(\)](#), [initializeCVODEobject\(\)](#), [initializePatchwork\(\)](#), [setDiscreteDimensionsOfLattice\(\)](#), [setPhysicalDimensionsOfLattice\(\)](#), [Simulation\(\)](#), [start\(\)](#), and [~Simulation\(\)](#).

#### 5.16.4.8 t

`sunrealtype Simulation::t [private]`

current time of the simulation

Definition at line 37 of file [SimulationClass.h](#).

Referenced by [advanceToTime\(\)](#), and [Simulation\(\)](#).

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

- src/[SimulationClass.h](#)
- src/[SimulationClass.cpp](#)

## 5.17 TimeEvolution Class Reference

monostate `TimeEvolution` class to propagate the field data in time in a given order of the HE weak-field expansion

```
#include <src/TimeEvolutionFunctions.h>
```

### Static Public Member Functions

- static int `f` (`sunrealtype t, N_Vector u, N_Vector udot, void *data_loc`)  
*CVODE right hand side function (CVRhsFn) to provide IVP of the ODE.*

## Static Public Attributes

- static int \* **c** = nullptr  
*choice which processes of the weak field expansion are included*
- static void(\* **TimeEvolver** )(LatticePatch \*, N\_Vector, N\_Vector, int \*) = **nonlinear1DProp**  
*Pointer to functions for differentiation and time evolution.*

### 5.17.1 Detailed Description

monostate [TimeEvolution](#) class to propagate the field data in time in a given order of the HE weak-field expansion

Definition at line 15 of file [TimeEvolutionFunctions.h](#).

### 5.17.2 Member Function Documentation

#### 5.17.2.1 f()

```
int TimeEvolution::f (
    sunrealtype t,
    N_Vector u,
    N_Vector udot,
    void * data_loc ) [static]
```

CVODE right hand side function (CVRhsFn) to provide IVP of the ODE.

CVode right-hand-side function (CVRhsFn)

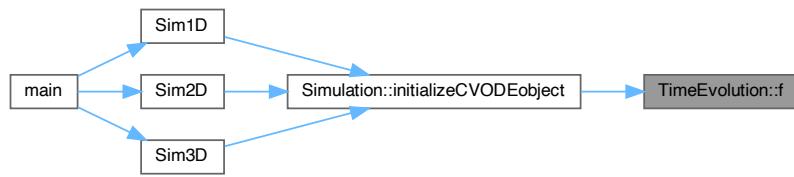
Definition at line 11 of file [TimeEvolutionFunctions.cpp](#).

```
00011
00012
00013 // Set recover pointer to provided lattice patch where the field data resides
00014 LatticePatch *data = static_cast<LatticePatch *>(data_loc);
00015
00016 // update circle
00017 // Access provided field values and temp. derivatievees with NVector pointers
00018 #if defined(_OPENMP)
00019     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00020             *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00021 #else
00022     sunrealtype *udata = NV_DATA_P(u),
00023             *dudata = NV_DATA_P(udot);
00024 #endif
00025
00026 // Store original data location of the patch
00027 sunrealtype *originaluData = data->uData,
00028         *originalduData = data->duData;
00029
00030 // Point patch data to arguments of f
00031 data->uData = udata;
00032 data->duData = dudata;
00033
00034 // Time-evolve these arguments (the field data) with specific propagator below
00035 TimeEvolver(data, u, udot, c);
00036
00037 // Refer patch data back to original location
00038 data->uData = originaluData;
00039 data->duData = originalduData;
00040
00041 return (0);
00042 }
```

References [c](#), [LatticePatch::duData](#), [TimeEvolver](#), and [LatticePatch::uData](#).

Referenced by [Simulation::initializeCVODEobject\(\)](#).

Here is the caller graph for this function:



### 5.17.3 Field Documentation

#### 5.17.3.1 c

```
int * TimeEvolution::c = nullptr [static]
```

choice which processes of the weak field expansion are included

Definition at line 18 of file [TimeEvolutionFunctions.h](#).

Referenced by [f\(\)](#), [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

#### 5.17.3.2 TimeEvolver

```
void(* TimeEvolution::TimeEvolver) (LatticePatch *, N\_Vecotr, N\_Vecotr, int *) = nonlinear1DProp [static]
```

Pointer to functions for differentiation and time evolution.

Definition at line 21 of file [TimeEvolutionFunctions.h](#).

Referenced by [f\(\)](#), [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

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

- [src/TimeEvolutionFunctions.h](#)
- [src/SimulationFunctions.cpp](#)
- [src/TimeEvolutionFunctions.cpp](#)



# Chapter 6

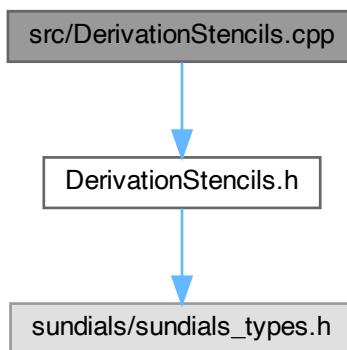
## File Documentation

### 6.1 README.md File Reference

### 6.2 src/DerivationStencils.cpp File Reference

Empty. All definitions in the header.

```
#include "DerivationStencils.h"  
Include dependency graph for DerivationStencils.cpp:
```



#### 6.2.1 Detailed Description

Empty. All definitions in the header.

Definition in file [DerivationStencils.cpp](#).

## 6.3 DerivationStencils.cpp

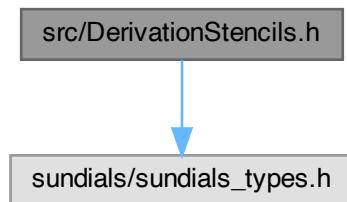
[Go to the documentation of this file.](#)

```
00001 ///////////////////////////////////////////////////////////////////
00002 /// @file DerivationStencils.cpp
00003 /// @brief Empty. All definitions in the header.
00004 ///////////////////////////////////////////////////////////////////
00005 #include "DerivationStencils.h"
```

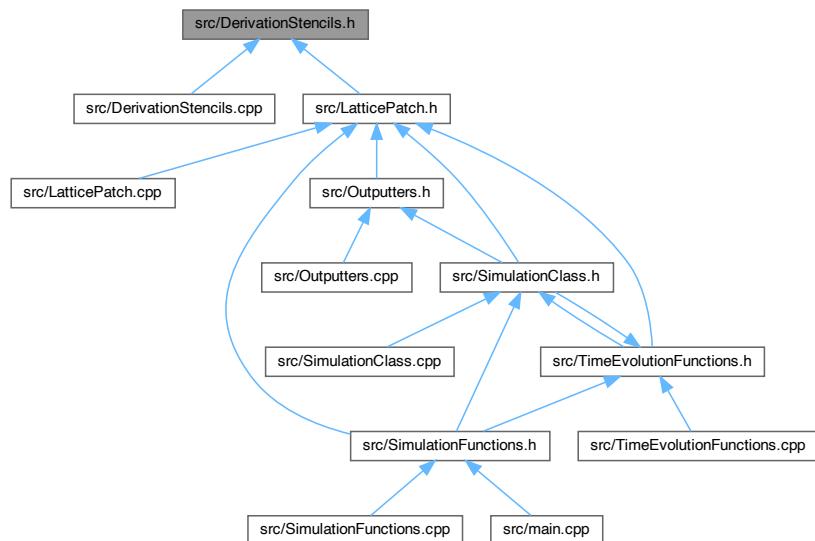
## 6.4 src/DerivationStencils.h File Reference

Definition of derivation stencils from order 1 to 13.

```
#include <sundials/sundials_types.h>
Include dependency graph for DerivationStencils.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- sunrealtype **s1f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s1b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s2f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s2c** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s2b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s3f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s3b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s4f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s4c** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s4b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s5f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s5b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s6f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s6c** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s6b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s7f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s7b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s8f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s8c** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s8b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s9f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s9b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s10f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s10c** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s10b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s11f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s11b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s12f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s12c** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s12b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s13f** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s13b** (sunrealtype const \*udata, const int dPD)
- sunrealtype **s1f** (sunrealtype const \*udata)
- sunrealtype **s1b** (sunrealtype const \*udata)
- sunrealtype **s2f** (sunrealtype const \*udata)
- sunrealtype **s2c** (sunrealtype const \*udata)
- sunrealtype **s2b** (sunrealtype const \*udata)
- sunrealtype **s3f** (sunrealtype const \*udata)
- sunrealtype **s3b** (sunrealtype const \*udata)
- sunrealtype **s4f** (sunrealtype const \*udata)
- sunrealtype **s4c** (sunrealtype const \*udata)
- sunrealtype **s4b** (sunrealtype const \*udata)
- sunrealtype **s5f** (sunrealtype const \*udata)
- sunrealtype **s5b** (sunrealtype const \*udata)
- sunrealtype **s6f** (sunrealtype const \*udata)
- sunrealtype **s6c** (sunrealtype const \*udata)
- sunrealtype **s6b** (sunrealtype const \*udata)
- sunrealtype **s7f** (sunrealtype const \*udata)
- sunrealtype **s7b** (sunrealtype const \*udata)
- sunrealtype **s8f** (sunrealtype const \*udata)
- sunrealtype **s8c** (sunrealtype const \*udata)
- sunrealtype **s8b** (sunrealtype const \*udata)
- sunrealtype **s9f** (sunrealtype const \*udata)

- sunrealtype [s9b](#) (sunrealtype const \*udata)
- sunrealtype [s10f](#) (sunrealtype const \*udata)
- sunrealtype [s10c](#) (sunrealtype const \*udata)
- sunrealtype [s10b](#) (sunrealtype const \*udata)
- sunrealtype [s11f](#) (sunrealtype const \*udata)
- sunrealtype [s11b](#) (sunrealtype const \*udata)
- sunrealtype [s12f](#) (sunrealtype const \*udata)
- sunrealtype [s12c](#) (sunrealtype const \*udata)
- sunrealtype [s12b](#) (sunrealtype const \*udata)
- sunrealtype [s13f](#) (sunrealtype const \*udata)
- sunrealtype [s13b](#) (sunrealtype const \*udata)

#### 6.4.1 Detailed Description

Definition of derivation stencils from order 1 to 13.

Definition in file [DerivationStencils.h](#).

#### 6.4.2 Function Documentation

##### 6.4.2.1 [s10b\(\)](#) [1/2]

```
sunrealtype s10b (
    unrealtype const * udata ) [inline]
```

Definition at line 275 of file [DerivationStencils.h](#).  
00276 { [return s10b\(udata, 6\);](#) }

References [s10b\(\)](#).

Here is the call graph for this function:



### 6.4.2.2 s10b() [2/2]

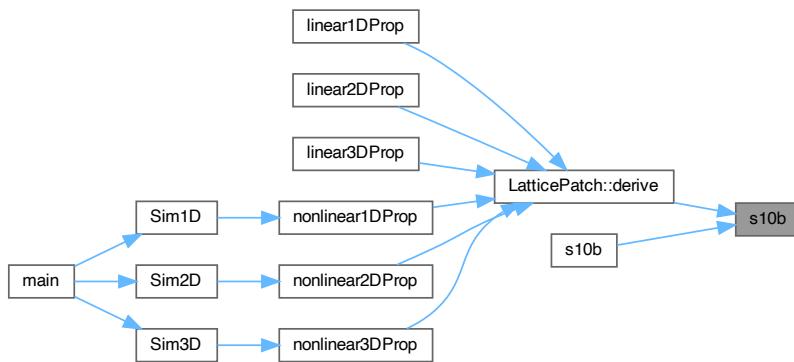
```
sunrealtype s10b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 169 of file [DerivationStencils.h](#).

```
00169
00170     return 1.0 / 840.0 * udata[-4 * dPD] - 1.0 / 63.0 * udata[-3 * dPD] +
00171         3.0 / 28.0 * udata[-2 * dPD] - 4.0 / 7.0 * udata[-1 * dPD] -
00172         11.0 / 30.0 * udata[0] + 6.0 / 5.0 * udata[1 * dPD] -
00173         1.0 / 2.0 * udata[2 * dPD] + 4.0 / 21.0 * udata[3 * dPD] -
00174         3.0 / 56.0 * udata[4 * dPD] + 1.0 / 105.0 * udata[5 * dPD] -
00175         1.0 / 1260.0 * udata[6 * dPD];
00176 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s10b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.3 s10c() [1/2]

```
sunrealtype s10c (
    unrealtype const * udata ) [inline]
```

Definition at line 273 of file [DerivationStencils.h](#).

```
00274 { return s10c(udata, 6); }
```

References [s10c\(\)](#).

Here is the call graph for this function:



#### 6.4.2.4 s10c() [2/2]

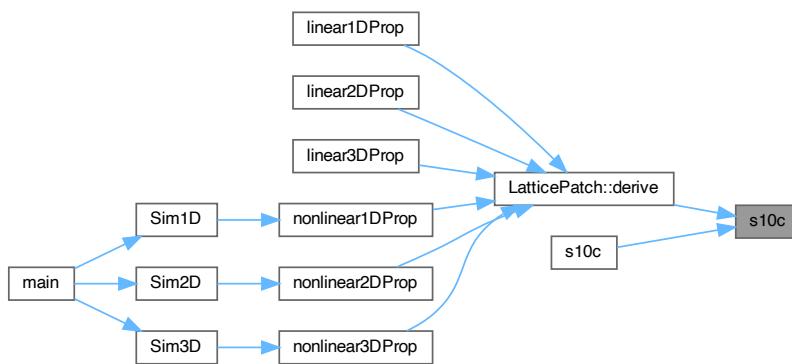
```
sunrealtype s10c (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 161 of file [DerivationStencils.h](#).

```
00161     {
00162     return -1.0 / 1260.0 * udata[-5 * dPD] + 5.0 / 504.0 * udata[-4 * dPD] -
00163         5.0 / 84.0 * udata[-3 * dPD] + 5.0 / 21.0 * udata[-2 * dPD] -
00164         5.0 / 6.0 * udata[-1 * dPD] + 0 + 5.0 / 6.0 * udata[1 * dPD] -
00165         5.0 / 21.0 * udata[2 * dPD] + 5.0 / 84.0 * udata[3 * dPD] -
00166         5.0 / 504.0 * udata[4 * dPD] + 1.0 / 1260.0 * udata[5 * dPD];
00167 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s10c\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.5 s10f() [1/2]

```
sunrealtype s10f (
    unrealtype const * udata ) [inline]
```

Definition at line 271 of file [DerivationStencils.h](#).

```
00272 { return s10f(udata, 6); }
```

References [s10f\(\)](#).

Here is the call graph for this function:



### 6.4.2.6 s10f() [2/2]

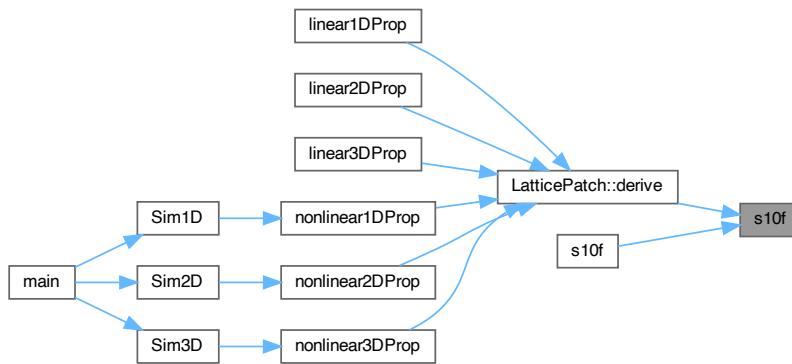
```
sunrealtype s10f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 152 of file [DerivationStencils.h](#).

```
00152
00153     return 1.0 / 1260.0 * udata[-6 * dPD] - 1.0 / 105.0 * udata[-5 * dPD] +
00154         3.0 / 56.0 * udata[-4 * dPD] - 4.0 / 21.0 * udata[-3 * dPD] +
00155         1.0 / 2.0 * udata[-2 * dPD] - 6.0 / 5.0 * udata[-1 * dPD] +
00156         11.0 / 30.0 * udata[0] + 4.0 / 7.0 * udata[1 * dPD] -
00157         3.0 / 28.0 * udata[2 * dPD] + 1.0 / 63.0 * udata[3 * dPD] -
00158         1.0 / 840.0 * udata[4 * dPD];
00159 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s10f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.7 s11b() [1/2]

```
sunrealtype s11b (
    unrealtype const * udata ) [inline]
```

Definition at line 279 of file [DerivationStencils.h](#).

```
00280 { return s11b(udata, 6); }
```

References [s11b\(\)](#).

Here is the call graph for this function:



### 6.4.2.8 s11b() [2/2]

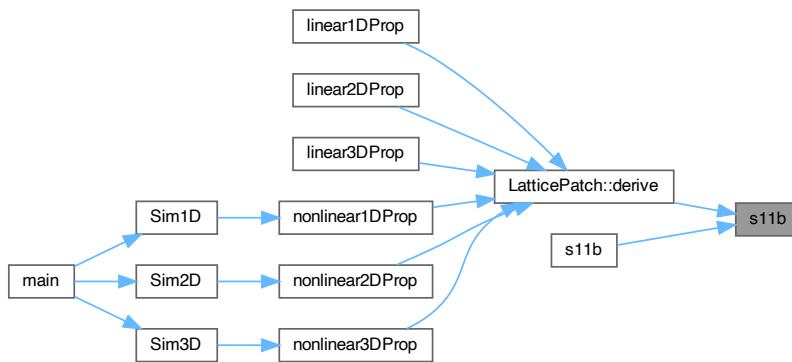
```
sunrealtype s11b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 187 of file [DerivationStencils.h](#).

```
00187
00188     return -1.0 / 2310.0 * udata[-5 * dPD] + 1.0 / 168.0 * udata[-4 * dPD] -
00189         5.0 / 126.0 * udata[-3 * dPD] + 5.0 / 28.0 * udata[-2 * dPD] -
00190         5.0 / 7.0 * udata[-1 * dPD] - 1.0 / 6.0 * udata[0] + udata[1 * dPD] -
00191         5.0 / 14.0 * udata[2 * dPD] + 5.0 / 42.0 * udata[3 * dPD] -
00192         5.0 / 168.0 * udata[4 * dPD] + 1.0 / 210.0 * udata[5 * dPD] -
00193         1.0 / 2772.0 * udata[6 * dPD];
00194 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s11b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.9 s11f() [1/2]

```
sunrealtype s11f (
    unrealtype const * udata ) [inline]
```

Definition at line 277 of file [DerivationStencils.h](#).

```
00278 { return s11f(udata, 6); }
```

References [s11f\(\)](#).

Here is the call graph for this function:



### 6.4.2.10 s11f() [2/2]

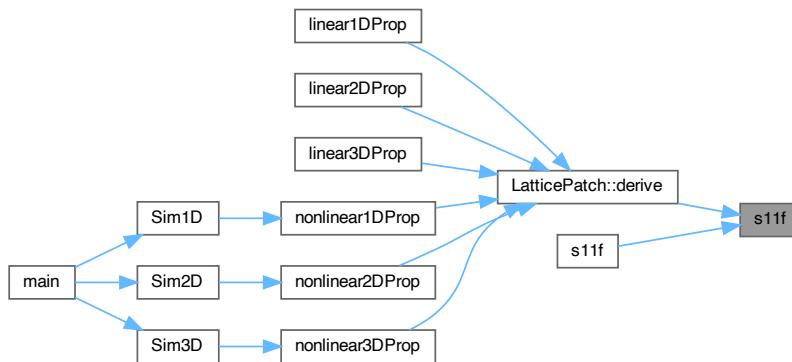
```
sunrealtype s11f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 178 of file [DerivationStencils.h](#).

```
00178
00179     return 1.0 / 2772.0 * udata[-6 * dPD] - 1.0 / 210.0 * udata[-5 * dPD] +
00180         5.0 / 168.0 * udata[-4 * dPD] - 5.0 / 42.0 * udata[-3 * dPD] +
00181         5.0 / 14.0 * udata[-2 * dPD] - 1.0 / 1.0 * udata[-1 * dPD] +
00182         1.0 / 6.0 * udata[0] + 5.0 / 7.0 * udata[1 * dPD] -
00183         5.0 / 28.0 * udata[2 * dPD] + 5.0 / 126.0 * udata[3 * dPD] -
00184         1.0 / 168.0 * udata[4 * dPD] + 1.0 / 2310.0 * udata[5 * dPD];
00185 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s11f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.11 s12b() [1/2]

```
sunrealtype s12b (
    unrealtype const * udata ) [inline]
```

Definition at line 285 of file [DerivationStencils.h](#).

```
00286 { return s12b(udata, 6); }
```

References [s12b\(\)](#).

Here is the call graph for this function:



### 6.4.2.12 s12b() [2/2]

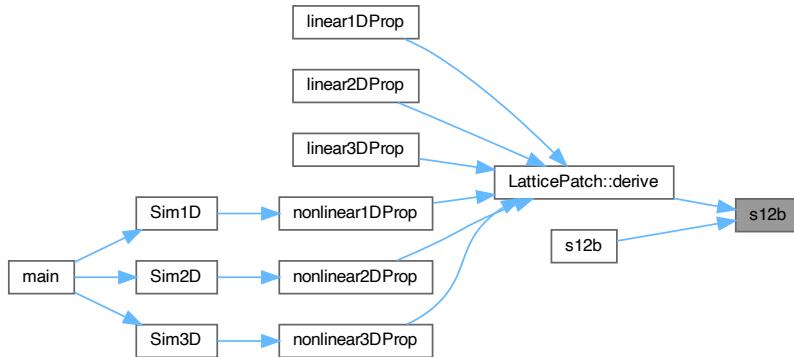
```
sunrealtype s12b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 215 of file [DerivationStencils.h](#).

```
00215
00216     return -1.0 / 3960.0 * udata[-5 * dPD] + 1.0 / 264.0 * udata[-4 * dPD] -
00217         1.0 / 36.0 * udata[-3 * dPD] + 5.0 / 36.0 * udata[-2 * dPD] -
00218         5.0 / 8.0 * udata[-1 * dPD] - 13.0 / 42.0 * udata[0] +
00219         7.0 / 6.0 * udata[1 * dPD] - 1.0 / 2.0 * udata[2 * dPD] +
00220         5.0 / 24.0 * udata[3 * dPD] - 5.0 / 72.0 * udata[4 * dPD] +
00221         1.0 / 60.0 * udata[5 * dPD] - 1.0 / 396.0 * udata[6 * dPD] +
00222         1.0 / 5544.0 * udata[7 * dPD];
00223 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s12b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.13 s12c() [1/2]

```
sunrealtype s12c (
    unrealtype const * udata ) [inline]
```

Definition at line 283 of file [DerivationStencils.h](#).

```
00284 { return s12c(udata, 6); }
```

References [s12c\(\)](#).

Here is the call graph for this function:



#### 6.4.2.14 s12c() [2/2]

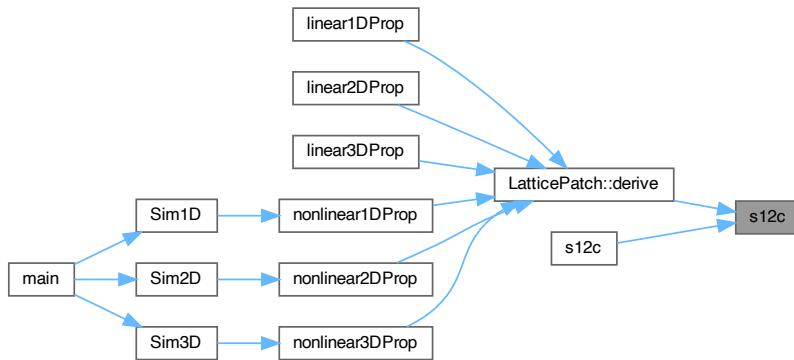
```
sunrealtype s12c (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 206 of file [DerivationStencils.h](#).

```
00206
00207     return 1.0 / 5544.0 * udata[-6 * dPD] - 1.0 / 385.0 * udata[-5 * dPD] +
00208         1.0 / 56.0 * udata[-4 * dPD] - 5.0 / 63.0 * udata[-3 * dPD] +
00209         15.0 / 56.0 * udata[-2 * dPD] - 6.0 / 7.0 * udata[-1 * dPD] + 0 +
00210         6.0 / 7.0 * udata[1 * dPD] - 15.0 / 56.0 * udata[2 * dPD] +
00211         5.0 / 63.0 * udata[3 * dPD] - 1.0 / 56.0 * udata[4 * dPD] +
00212         1.0 / 385.0 * udata[5 * dPD] - 1.0 / 5544.0 * udata[6 * dPD];
00213 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s12c\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.15 s12f() [1/2]

```
sunrealtype s12f (
    unrealtype const * udata ) [inline]
```

Definition at line 281 of file [DerivationStencils.h](#).

```
00282 { return s12f(udata, 6); }
```

References [s12f\(\)](#).

Here is the call graph for this function:



#### 6.4.2.16 s12f() [2/2]

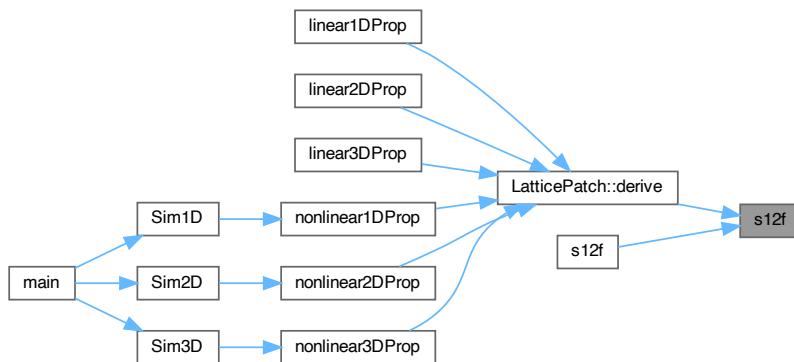
```
sunrealtype s12f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 196 of file [DerivationStencils.h](#).

```
00196
00197     return -1.0 / 5544.0 * udata[-7 * dPD] + 1.0 / 396.0 * udata[-6 * dPD] -
00198         1.0 / 60.0 * udata[-5 * dPD] + 5.0 / 72.0 * udata[-4 * dPD] -
00199         5.0 / 24.0 * udata[-3 * dPD] + 1.0 / 2.0 * udata[-2 * dPD] -
00200         7.0 / 6.0 * udata[-1 * dPD] + 13.0 / 42.0 * udata[0] +
00201         5.0 / 8.0 * udata[1 * dPD] - 5.0 / 36.0 * udata[2 * dPD] +
00202         1.0 / 36.0 * udata[3 * dPD] - 1.0 / 264.0 * udata[4 * dPD] +
00203         1.0 / 3960.0 * udata[5 * dPD];
00204 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s12f\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.17 s13b() [1/2]

```
sunrealtype s13b (
    unrealtype const * udata ) [inline]
```

Definition at line 289 of file [DerivationStencils.h](#).

```
00290 { return s13b(udata, 6); }
```

References [s13b\(\)](#).

Here is the call graph for this function:



### 6.4.2.18 s13b() [2/2]

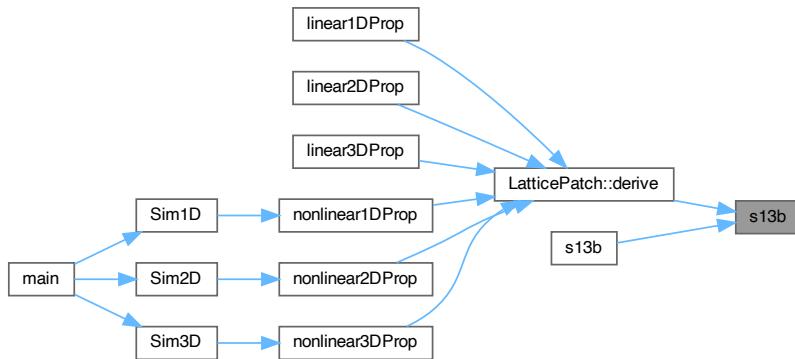
```
sunrealtype s13b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 235 of file [DerivationStencils.h](#).

```
00235
00236     return 1.0 / 10296.0 * udata[-6 * dPD] - 1.0 / 660.0 * udata[-5 * dPD] +
00237         1.0 / 88.0 * udata[-4 * dPD] - 1.0 / 18.0 * udata[-3 * dPD] +
00238         5.0 / 24.0 * udata[-2 * dPD] - 3.0 / 4.0 * udata[-1 * dPD] -
00239         1.0 / 7.0 * udata[0] + udata[1 * dPD] - 3.0 / 8.0 * udata[2 * dPD] +
00240         5.0 / 36.0 * udata[3 * dPD] - 1.0 / 24.0 * udata[4 * dPD] +
00241         1.0 / 110.0 * udata[5 * dPD] - 1.0 / 792.0 * udata[6 * dPD] +
00242         1.0 / 12012.0 * udata[7 * dPD];
00243 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s13b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.19 s13f() [1/2]

```
sunrealtype s13f (
    unrealtype const * udata ) [inline]
```

Definition at line 287 of file [DerivationStencils.h](#).

```
00288 { return s13f(udata, 6); }
```

References [s13f\(\)](#).

Here is the call graph for this function:



### 6.4.2.20 s13f() [2/2]

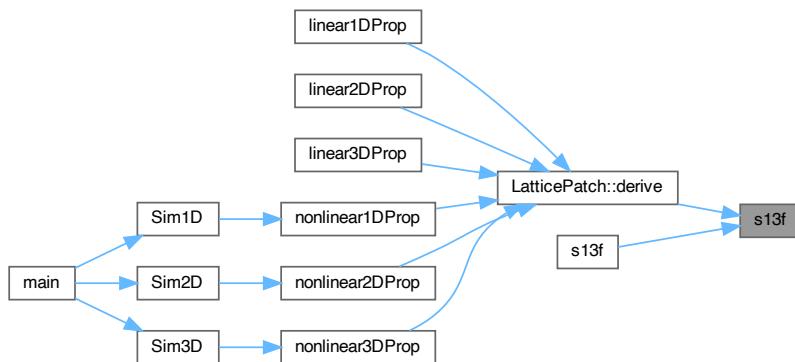
```
sunrealtype s13f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 225 of file [DerivationStencils.h](#).

```
00225
00226     return -1.0 / 12012.0 * udata[-7 * dPD] + 1.0 / 792.0 * udata[-6 * dPD] -
00227         1.0 / 110.0 * udata[-5 * dPD] + 1.0 / 24.0 * udata[-4 * dPD] -
00228         5.0 / 36.0 * udata[-3 * dPD] + 3.0 / 8.0 * udata[-2 * dPD] -
00229         1.0 / 1.0 * udata[-1 * dPD] + 1.0 / 7.0 * udata[0] +
00230         3.0 / 4.0 * udata[1 * dPD] - 5.0 / 24.0 * udata[2 * dPD] +
00231         1.0 / 18.0 * udata[3 * dPD] - 1.0 / 88.0 * udata[4 * dPD] +
00232         1.0 / 660.0 * udata[5 * dPD] - 1.0 / 10296.0 * udata[6 * dPD];
00233 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s13f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.21 s1b() [1/2]

```
sunrealtype s1b (
    unrealtype const * udata ) [inline]
```

Definition at line 250 of file [DerivationStencils.h](#).

```
00250 { return s1b(udata, 6); }
```

References [s1b\(\)](#).

Here is the call graph for this function:



### 6.4.2.22 s1b() [2/2]

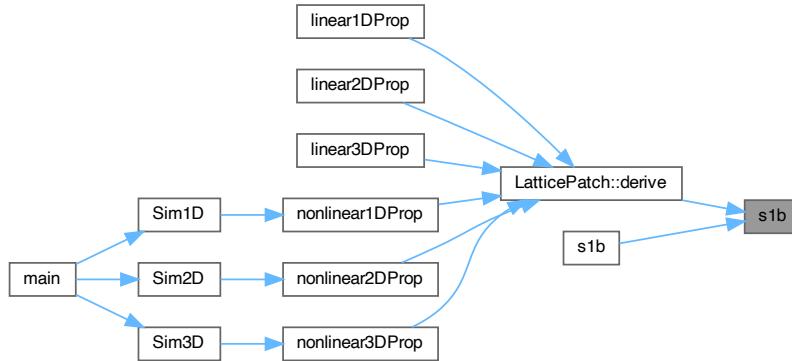
```
sunrealtype s1b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 21 of file [DerivationStencils.h](#).

```
00021
00022     return -1.0 / 1.0 * udata[0] + udata[1 * dPD];
00023 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s1b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.23 s1f() [1/2]

```
sunrealtype s1f (
    unrealtype const * udata ) [inline]
```

Definition at line 249 of file [DerivationStencils.h](#).

```
00249 { return s1f(udata, 6); }
```

References [s1f\(\)](#).

Here is the call graph for this function:



### 6.4.2.24 s1f() [2/2]

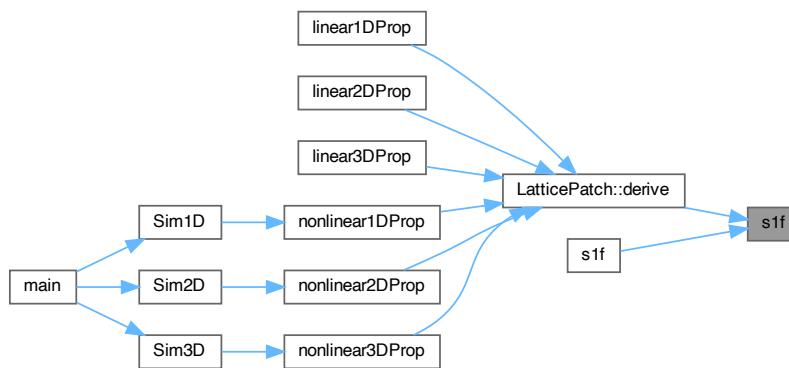
```
sunrealtype s1f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 16 of file [DerivationStencils.h](#).

```
00016
00017   return -1.0 / 1.0 * udata[-1 * dPD] + udata[0];
00018 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s1f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.25 s2b() [1/2]

```
sunrealtype s2b (
    unrealtype const * udata ) [inline]
```

Definition at line 253 of file [DerivationStencils.h](#).

```
00253 { return s2b(udata, 6); }
```

References [s2b\(\)](#).

Here is the call graph for this function:



### 6.4.2.26 s2b() [2/2]

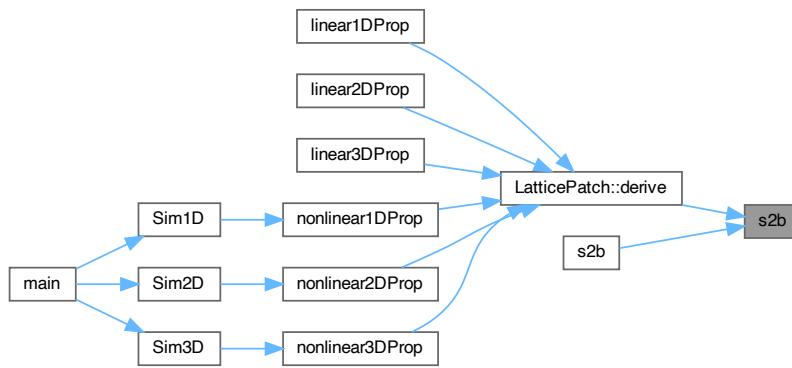
```
sunrealtype s2b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 35 of file [DerivationStencils.h](#).

```
00035
00036     return -3.0 / 2.0 * udata[0] + 2.0 / 1.0 * udata[1 * dPD] -
00037         1.0 / 2.0 * udata[2 * dPD];
00038 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s2b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.27 s2c() [1/2]

```
sunrealtype s2c (
    unrealtype const * udata ) [inline]
```

Definition at line 252 of file [DerivationStencils.h](#).

```
00252 { return s2c(udata, 6); }
```

References [s2c\(\)](#).

Here is the call graph for this function:



### 6.4.2.28 s2c() [2/2]

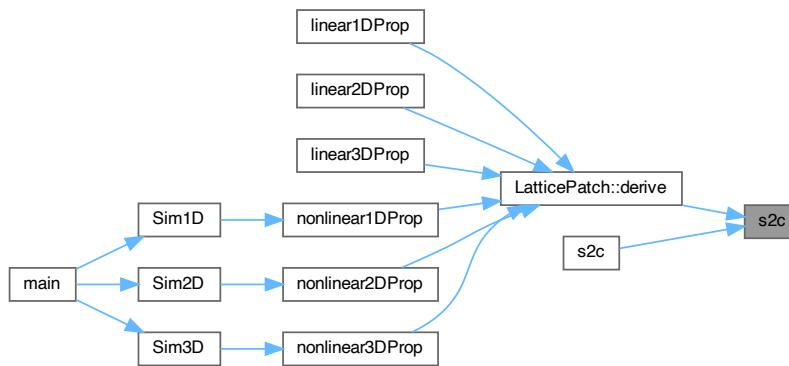
```
sunrealtype s2c (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 31 of file [DerivationStencils.h](#).

```
00031     {
00032     return -1.0 / 2.0 * udata[-1 * dPD] + 0 + 1.0 / 2.0 * udata[1 * dPD];
00033 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s2c\(\)](#).

Here is the caller graph for this function:



### 6.4.2.29 s2f() [1/2]

```
sunrealtype s2f (
    unrealtype const * udata ) [inline]
```

Definition at line 251 of file [DerivationStencils.h](#).

```
00251 { return s2f(udata, 6); }
```

References [s2f\(\)](#).

Here is the call graph for this function:



### 6.4.2.30 s2f() [2/2]

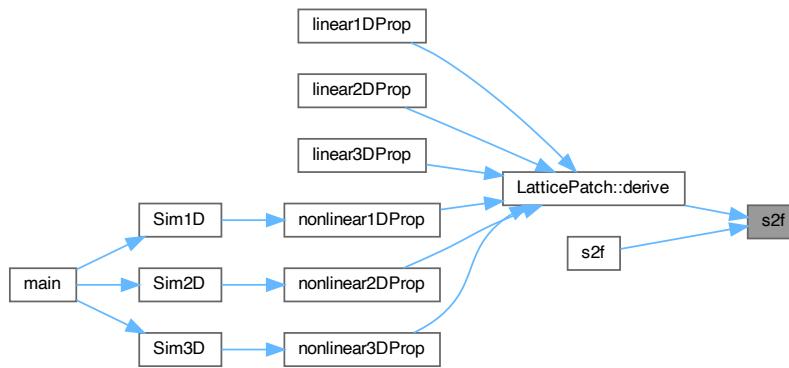
```
sunrealtype s2f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 26 of file [DerivationStencils.h](#).

```
00026     {
00027     return 1.0 / 2.0 * udata[-2 * dPD] - 2.0 / 1.0 * udata[-1 * dPD] +
00028         3.0 / 2.0 * udata[0];
00029 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s2f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.31 s3b() [1/2]

```
sunrealtype s3b (
    unrealtype const * udata ) [inline]
```

Definition at line 255 of file [DerivationStencils.h](#).

```
00255 { return s3b(udata, 6); }
```

References [s3b\(\)](#).

Here is the call graph for this function:



### 6.4.2.32 s3b() [2/2]

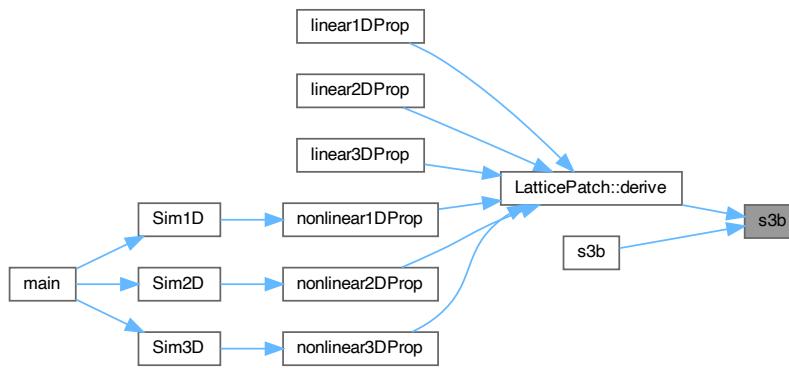
```
sunrealtype s3b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 45 of file [DerivationStencils.h](#).

```
00045     {
00046     return -1.0 / 3.0 * udata[-1 * dPD] - 1.0 / 2.0 * udata[0] + udata[1 * dPD] -
00047         1.0 / 6.0 * udata[2 * dPD];
00048 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s3b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.33 s3f() [1/2]

```
sunrealtype s3f (
    unrealtype const * udata ) [inline]
```

Definition at line 254 of file [DerivationStencils.h](#).

```
00254 { return s3f(udata, 6); }
```

References [s3f\(\)](#).

Here is the call graph for this function:



### 6.4.2.34 s3f() [2/2]

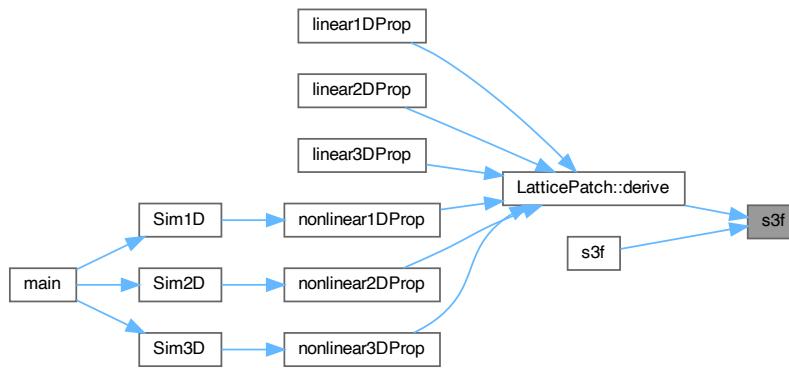
```
sunrealtype s3f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 40 of file [DerivationStencils.h](#).

```
00040
00041     return 1.0 / 6.0 * udata[-2 * dPD] - 1.0 / 1.0 * udata[-1 * dPD] +
00042             1.0 / 2.0 * udata[0] + 1.0 / 3.0 * udata[1 * dPD];
00043 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s3f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.35 s4b() [1/2]

```
sunrealtype s4b (
    unrealtype const * udata ) [inline]
```

Definition at line 258 of file [DerivationStencils.h](#).

```
00258 { return s4b(udata, 6); }
```

References [s4b\(\)](#).

Here is the call graph for this function:



### 6.4.2.36 s4b() [2/2]

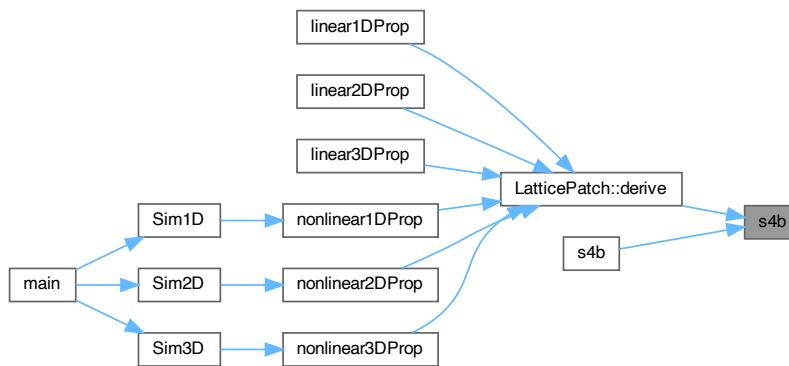
```
sunrealtype s4b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 61 of file [DerivationStencils.h](#).

```
00061
00062     return -1.0 / 4.0 * udata[-1 * dPD] - 5.0 / 6.0 * udata[0] +
00063         3.0 / 2.0 * udata[1 * dPD] - 1.0 / 2.0 * udata[2 * dPD] +
00064         1.0 / 12.0 * udata[3 * dPD];
00065 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s4b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.37 s4c() [1/2]

```
sunrealtype s4c (
    unrealtype const * udata ) [inline]
```

Definition at line 257 of file [DerivationStencils.h](#).

```
00257 { return s4c(udata, 6); }
```

References [s4c\(\)](#).

Here is the call graph for this function:



### 6.4.2.38 s4c() [2/2]

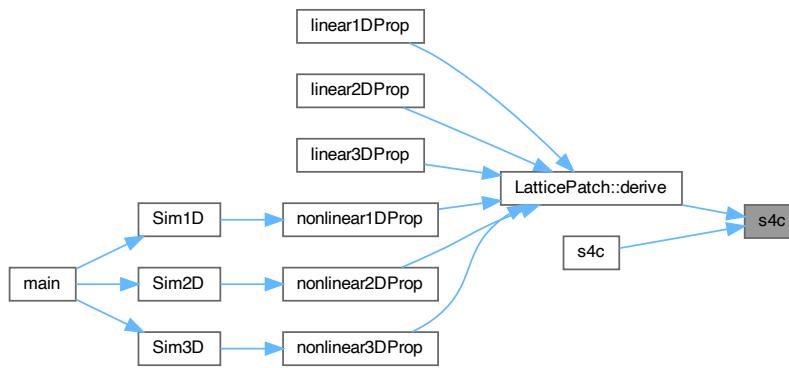
```
sunrealtype s4c (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 56 of file [DerivationStencils.h](#).

```
00056     {
00057     return 1.0 / 12.0 * udata[-2 * dPD] - 2.0 / 3.0 * udata[-1 * dPD] + 0 +
00058         2.0 / 3.0 * udata[1 * dPD] - 1.0 / 12.0 * udata[2 * dPD];
00059 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s4c\(\)](#).

Here is the caller graph for this function:



### 6.4.2.39 s4f() [1/2]

```
sunrealtype s4f (
    unrealtype const * udata ) [inline]
```

Definition at line 256 of file [DerivationStencils.h](#).

```
00256 { return s4f(udata, 6); }
```

References [s4f\(\)](#).

Here is the call graph for this function:



#### 6.4.2.40 s4f() [2/2]

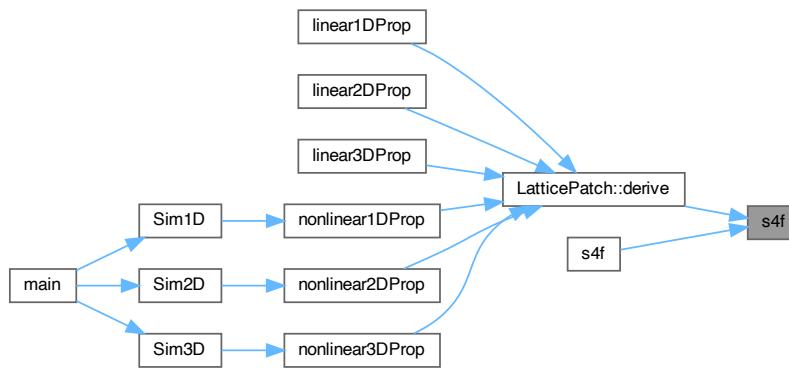
```
sunrealtype s4f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 50 of file [DerivationStencils.h](#).

```
00050
00051     return -1.0 / 12.0 * udata[-3 * dPD] + 1.0 / 2.0 * udata[-2 * dPD] -
00052         3.0 / 2.0 * udata[-1 * dPD] + 5.0 / 6.0 * udata[0] +
00053         1.0 / 4.0 * udata[1 * dPD];
00054 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s4f\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.41 s5b() [1/2]

```
sunrealtype s5b (
    unrealtype const * udata ) [inline]
```

Definition at line 260 of file [DerivationStencils.h](#).

```
00260 { return s5b(udata, 6); }
```

References [s5b\(\)](#).

Here is the call graph for this function:



#### 6.4.2.42 s5b() [2/2]

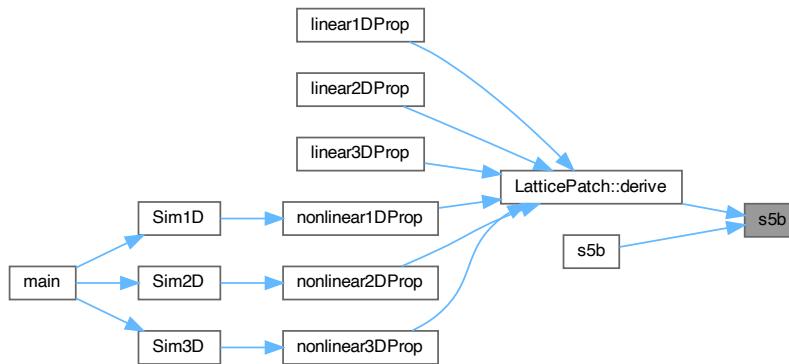
```
sunrealtype s5b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 73 of file [DerivationStencils.h](#).

```
00073     {
00074     return 1.0 / 20.0 * udata[-2 * dPD] - 1.0 / 2.0 * udata[-1 * dPD] -
00075         1.0 / 3.0 * udata[0] + udata[1 * dPD] - 1.0 / 4.0 * udata[2 * dPD] +
00076         1.0 / 30.0 * udata[3 * dPD];
00077 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s5b\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.43 s5f() [1/2]

```
sunrealtype s5f (
    unrealtype const * udata ) [inline]
```

Definition at line 259 of file [DerivationStencils.h](#).

```
00259 { return s5f(udata, 6); }
```

References [s5f\(\)](#).

Here is the call graph for this function:



#### 6.4.2.44 s5f() [2/2]

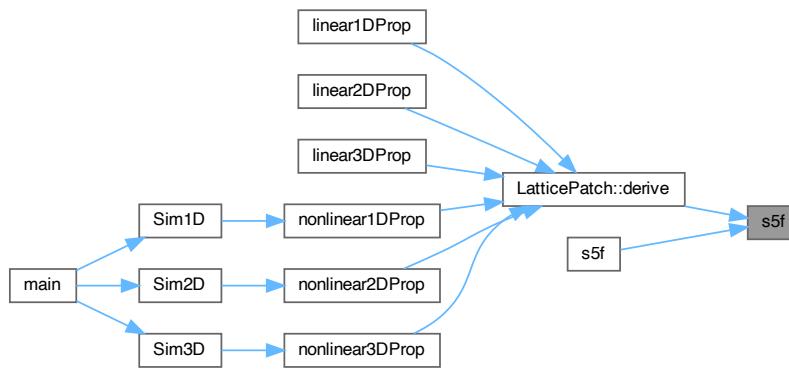
```
sunrealtype s5f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 67 of file [DerivationStencils.h](#).

```
00067
00068     return -1.0 / 30.0 * udata[-3 * dPD] + 1.0 / 4.0 * udata[-2 * dPD] -
00069         1.0 / 1.0 * udata[-1 * dPD] + 1.0 / 3.0 * udata[0] +
00070         1.0 / 2.0 * udata[1 * dPD] - 1.0 / 20.0 * udata[2 * dPD];
00071 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s5f\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.45 s6b() [1/2]

```
sunrealtype s6b (
    unrealtype const * udata ) [inline]
```

Definition at line 263 of file [DerivationStencils.h](#).

```
00263 { return s6b(udata, 6); }
```

References [s6b\(\)](#).

Here is the call graph for this function:



### 6.4.2.46 s6b() [2/2]

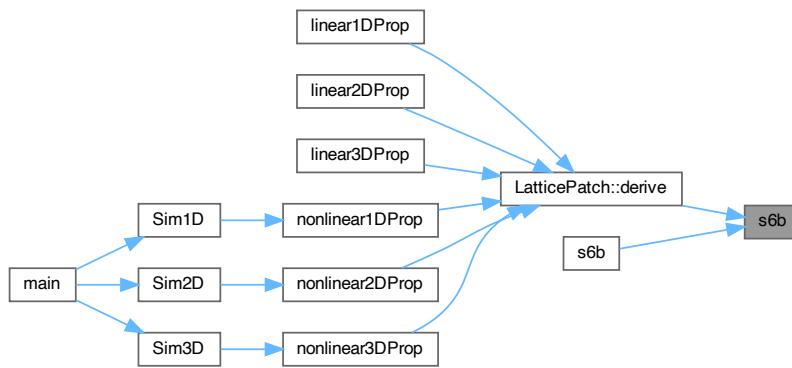
```
sunrealtype s6b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 92 of file [DerivationStencils.h](#).

```
00092     {
00093     return 1.0 / 30.0 * udata[-2 * dPD] - 2.0 / 5.0 * udata[-1 * dPD] -
00094         7.0 / 12.0 * udata[0] + 4.0 / 3.0 * udata[1 * dPD] -
00095         1.0 / 2.0 * udata[2 * dPD] + 2.0 / 15.0 * udata[3 * dPD] -
00096         1.0 / 60.0 * udata[4 * dPD];
00097 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s6b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.47 s6c() [1/2]

```
sunrealtype s6c (
    unrealtype const * udata ) [inline]
```

Definition at line 262 of file [DerivationStencils.h](#).

```
00262 { return s6c(udata, 6); }
```

References [s6c\(\)](#).

Here is the call graph for this function:



#### 6.4.2.48 s6c() [2/2]

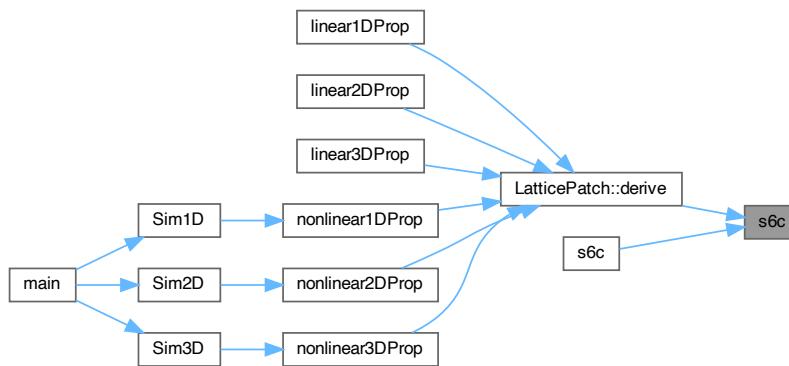
```
sunrealtype s6c (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 86 of file [DerivationStencils.h](#).

```
00086     {
00087     return -1.0 / 60.0 * udata[-3 * dPD] + 3.0 / 20.0 * udata[-2 * dPD] -
00088         3.0 / 4.0 * udata[-1 * dPD] + 0 + 3.0 / 4.0 * udata[1 * dPD] -
00089         3.0 / 20.0 * udata[2 * dPD] + 1.0 / 60.0 * udata[3 * dPD];
00090 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s6c\(\)](#).

Here is the caller graph for this function:



#### 6.4.2.49 s6f() [1/2]

```
sunrealtype s6f (
    unrealtype const * udata ) [inline]
```

Definition at line 261 of file [DerivationStencils.h](#).

```
00261 { return s6f(udata, 6); }
```

References [s6f\(\)](#).

Here is the call graph for this function:



### 6.4.2.50 s6f() [2/2]

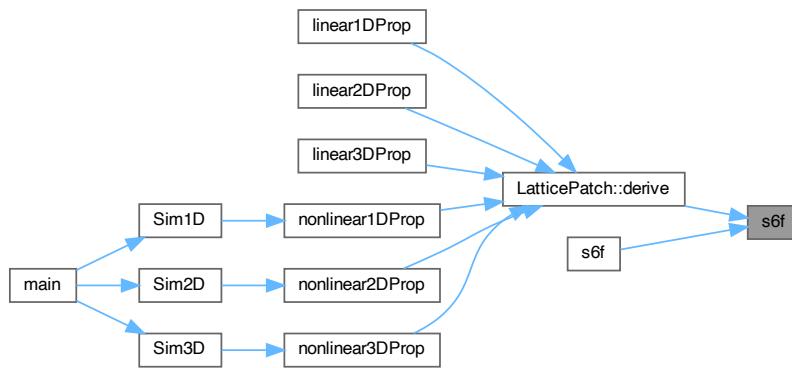
```
sunrealtype s6f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 79 of file [DerivationStencils.h](#).

```
00079
00080     return 1.0 / 60.0 * udata[-4 * dPD] - 2.0 / 15.0 * udata[-3 * dPD] +
00081         1.0 / 2.0 * udata[-2 * dPD] - 4.0 / 3.0 * udata[-1 * dPD] +
00082         7.0 / 12.0 * udata[0] + 2.0 / 5.0 * udata[1 * dPD] -
00083         1.0 / 30.0 * udata[2 * dPD];
00084 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s6f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.51 s7b() [1/2]

```
sunrealtype s7b (
    unrealtype const * udata ) [inline]
```

Definition at line 265 of file [DerivationStencils.h](#).

```
00265 { return s7b(udata, 6); }
```

References [s7b\(\)](#).

Here is the call graph for this function:



### 6.4.2.52 s7b() [2/2]

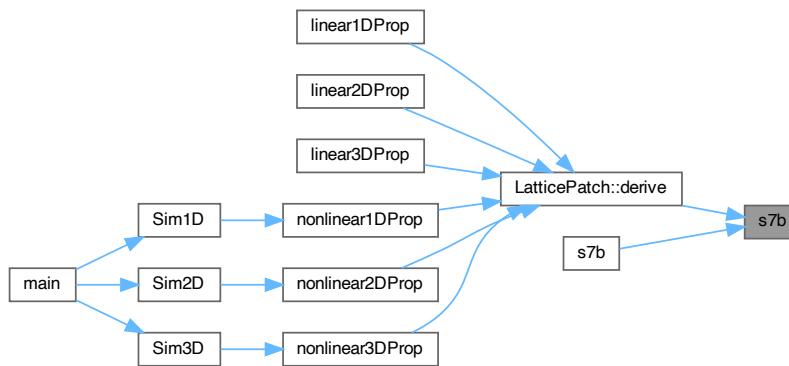
```
sunrealtype s7b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 106 of file [DerivationStencils.h](#).

```
00106     {
00107     return -1.0 / 105.0 * udata[-3 * dPD] + 1.0 / 10.0 * udata[-2 * dPD] -
00108         3.0 / 5.0 * udata[-1 * dPD] - 1.0 / 4.0 * udata[0] + udata[1 * dPD] -
00109         3.0 / 10.0 * udata[2 * dPD] + 1.0 / 15.0 * udata[3 * dPD] -
00110         1.0 / 140.0 * udata[4 * dPD];
00111 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s7b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.53 s7f() [1/2]

```
sunrealtype s7f (
    unrealtype const * udata ) [inline]
```

Definition at line 264 of file [DerivationStencils.h](#).

```
00264 { return s7f(udata, 6); }
```

References [s7f\(\)](#).

Here is the call graph for this function:



### 6.4.2.54 s7f() [2/2]

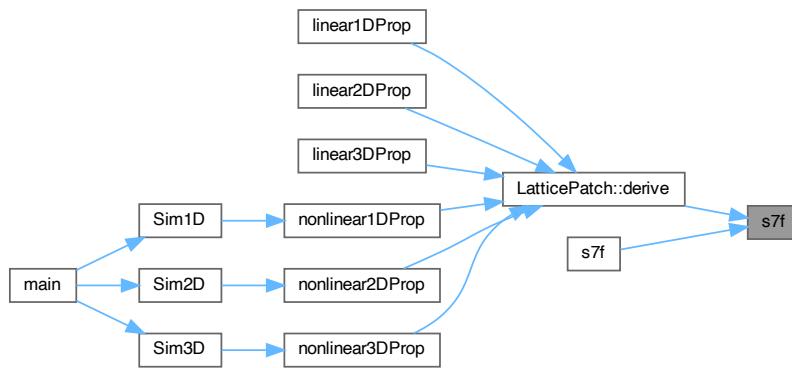
```
sunrealtype s7f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 99 of file [DerivationStencils.h](#).

```
00099     {
00100     return 1.0 / 140.0 * udata[-4 * dPD] - 1.0 / 15.0 * udata[-3 * dPD] +
00101         3.0 / 10.0 * udata[-2 * dPD] - 1.0 / 1.0 * udata[-1 * dPD] +
00102         1.0 / 4.0 * udata[0] + 3.0 / 5.0 * udata[1 * dPD] -
00103         1.0 / 10.0 * udata[2 * dPD] + 1.0 / 105.0 * udata[3 * dPD];
00104 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s7f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.55 s8b() [1/2]

```
sunrealtype s8b (
    unrealtype const * udata ) [inline]
```

Definition at line 268 of file [DerivationStencils.h](#).

```
00268 { return s8b(udata, 6); }
```

References [s8b\(\)](#).

Here is the call graph for this function:



### 6.4.2.56 s8b() [2/2]

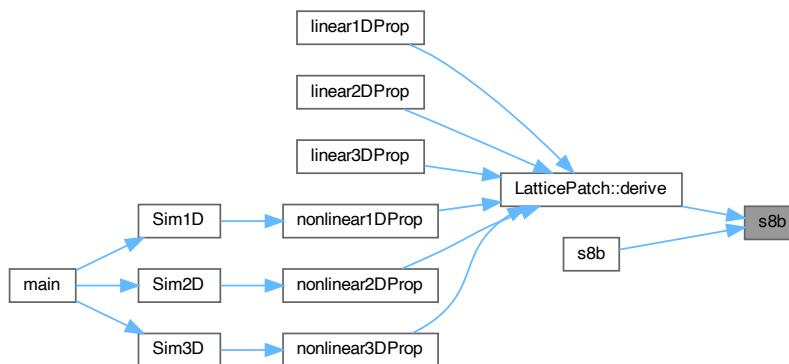
```
sunrealtype s8b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 128 of file [DerivationStencils.h](#).

```
00128     {
00129     return -1.0 / 168.0 * udata[-3 * dPD] + 1.0 / 14.0 * udata[-2 * dPD] -
00130         1.0 / 2.0 * udata[-1 * dPD] - 9.0 / 20.0 * udata[0] +
00131         5.0 / 4.0 * udata[1 * dPD] - 1.0 / 2.0 * udata[2 * dPD] +
00132         1.0 / 6.0 * udata[3 * dPD] - 1.0 / 28.0 * udata[4 * dPD] +
00133         1.0 / 280.0 * udata[5 * dPD];
00134 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s8b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.57 s8c() [1/2]

```
sunrealtype s8c (
    unrealtype const * udata ) [inline]
```

Definition at line 267 of file [DerivationStencils.h](#).

```
00267 { return s8c(udata, 6); }
```

References [s8c\(\)](#).

Here is the call graph for this function:



### 6.4.2.58 s8c() [2/2]

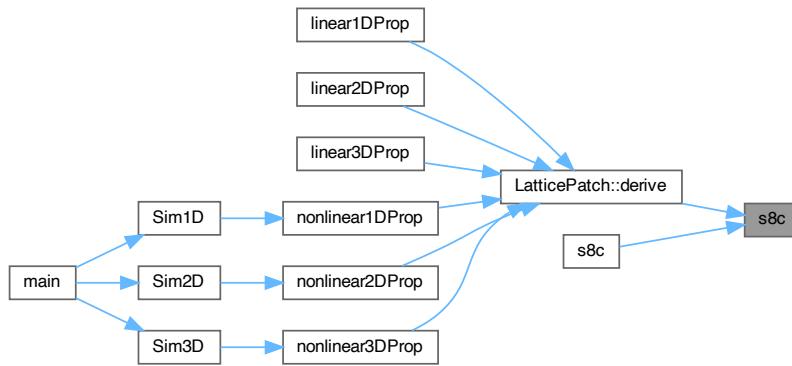
```
sunrealtype s8c (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 121 of file [DerivationStencils.h](#).

```
00121     {
00122     return 1.0 / 280.0 * udata[-4 * dPD] - 4.0 / 105.0 * udata[-3 * dPD] +
00123           1.0 / 5.0 * udata[-2 * dPD] - 4.0 / 5.0 * udata[-1 * dPD] + 0 +
00124           4.0 / 5.0 * udata[1 * dPD] - 1.0 / 5.0 * udata[2 * dPD] +
00125           4.0 / 105.0 * udata[3 * dPD] - 1.0 / 280.0 * udata[4 * dPD];
00126 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s8c\(\)](#).

Here is the caller graph for this function:



### 6.4.2.59 s8f() [1/2]

```
sunrealtype s8f (
    unrealtype const * udata ) [inline]
```

Definition at line 266 of file [DerivationStencils.h](#).

```
00266 { return s8f(udata, 6); }
```

References [s8f\(\)](#).

Here is the call graph for this function:



### 6.4.2.60 s8f() [2/2]

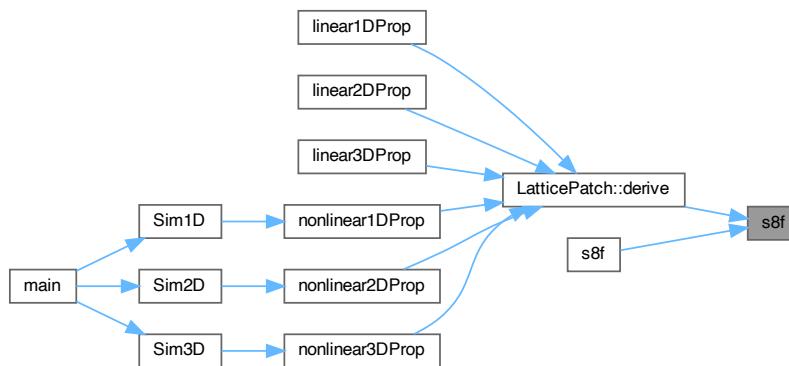
```
sunrealtype s8f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 113 of file [DerivationStencils.h](#).

```
00113     {
00114     return -1.0 / 280.0 * udata[-5 * dPD] + 1.0 / 28.0 * udata[-4 * dPD] -
00115         1.0 / 6.0 * udata[-3 * dPD] + 1.0 / 2.0 * udata[-2 * dPD] -
00116         5.0 / 4.0 * udata[-1 * dPD] + 9.0 / 20.0 * udata[0] +
00117         1.0 / 2.0 * udata[1 * dPD] - 1.0 / 14.0 * udata[2 * dPD] +
00118         1.0 / 168.0 * udata[3 * dPD];
00119 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s8f\(\)](#).

Here is the caller graph for this function:



### 6.4.2.61 s9b() [1/2]

```
sunrealtype s9b (
    unrealtype const * udata ) [inline]
```

Definition at line 270 of file [DerivationStencils.h](#).

```
00270 { return s9b(udata, 6); }
```

References [s9b\(\)](#).

Here is the call graph for this function:



### 6.4.2.62 s9b() [2/2]

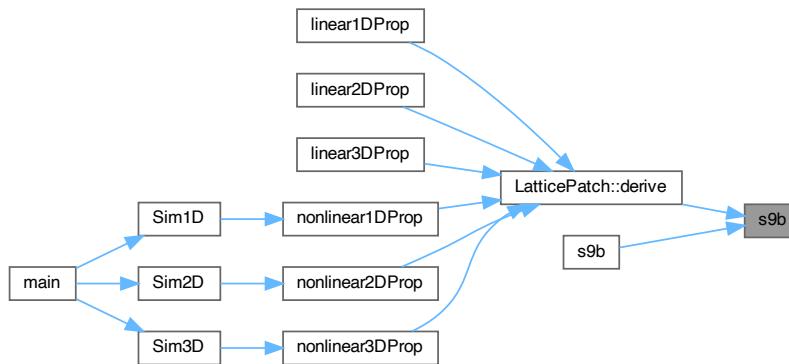
```
sunrealtype s9b (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 144 of file [DerivationStencils.h](#).

```
00144     {
00145     return 1.0 / 504.0 * udata[-4 * dPD] - 1.0 / 42.0 * udata[-3 * dPD] +
00146         1.0 / 7.0 * udata[-2 * dPD] - 2.0 / 3.0 * udata[-1 * dPD] -
00147         1.0 / 5.0 * udata[0] + udata[1 * dPD] - 1.0 / 3.0 * udata[2 * dPD] +
00148         2.0 / 21.0 * udata[3 * dPD] - 1.0 / 56.0 * udata[4 * dPD] +
00149         1.0 / 630.0 * udata[5 * dPD];
00150 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s9b\(\)](#).

Here is the caller graph for this function:



### 6.4.2.63 s9f() [1/2]

```
sunrealtype s9f (
    unrealtype const * udata ) [inline]
```

Definition at line 269 of file [DerivationStencils.h](#).

```
00269 { return s9f(udata, 6); }
```

References [s9f\(\)](#).

Here is the call graph for this function:



### 6.4.2.64 s9f() [2/2]

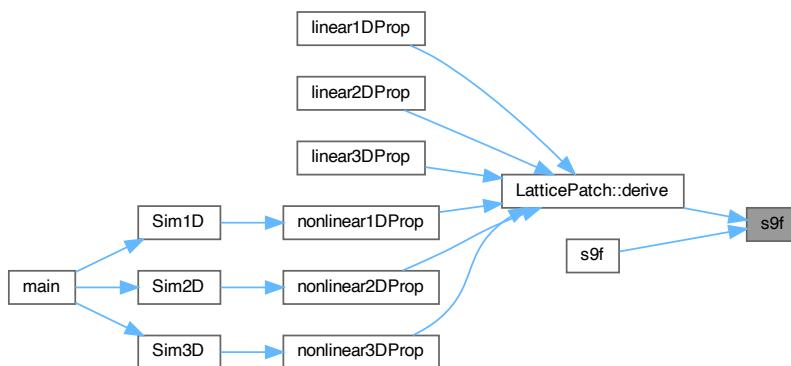
```
sunrealtype s9f (
    unrealtype const * udata,
    const int dPD ) [inline]
```

Definition at line 136 of file [DerivationStencils.h](#).

```
00136     {
00137     return -1.0 / 630.0 * udata[-5 * dPD] + 1.0 / 56.0 * udata[-4 * dPD] -
00138         2.0 / 21.0 * udata[-3 * dPD] + 1.0 / 3.0 * udata[-2 * dPD] -
00139         1.0 / 1.0 * udata[-1 * dPD] + 1.0 / 5.0 * udata[0] +
00140         2.0 / 3.0 * udata[1 * dPD] - 1.0 / 7.0 * udata[2 * dPD] +
00141         1.0 / 42.0 * udata[3 * dPD] - 1.0 / 504.0 * udata[4 * dPD];
00142 }
```

Referenced by [LatticePatch::derive\(\)](#), and [s9f\(\)](#).

Here is the caller graph for this function:



## 6.5 DerivationStencils.h

[Go to the documentation of this file.](#)

```
00001 ///////////////////////////////////////////////////////////////////
00002 /// @file DerivationStencils.h
00003 /// @brief Definition of derivation stencils from order 1 to 13
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #pragma once
00007
00008 #include <sundials/sundials_types.h> /* definition of type unrealtype */
00009
00010 ///////////////////////////////////////////////////////////////////
00011 // Stencils with variable dPD -- data point dimension //
00012 ///////////////////////////////////////////////////////////////////
00013
00014 // Downwind (forward) differentiating
00015 #pragma omp declare simd uniform(dPD) notinbranch
00016 inline unrealtype slf(unrealtype const *udata, const int dPD) {
00017     return -1.0 / 1.0 * udata[-1 * dPD] + udata[0];
00018 }
00019 // Upwind (backward) differentiating
00020 #pragma omp declare simd uniform(dPD) notinbranch
00021 inline unrealtype slb(unrealtype const *udata, const int dPD) {
00022     return -1.0 / 1.0 * udata[0] + udata[1 * dPD];
00023 }
00024
00025 #pragma omp declare simd uniform(dPD) notinbranch
00026 inline unrealtype s2f(unrealtype const *udata, const int dPD) {
00027     return 1.0 / 2.0 * udata[-2 * dPD] - 2.0 / 1.0 * udata[-1 * dPD] +
00028         1.0 / 2.0 * udata[0] + 2.0 / 1.0 * udata[1 * dPD];
00029 }
```

```

00028      3.0 / 2.0 * udata[0];
00029 }
00030 #pragma omp declare simd uniform(dPD) notinbranch
00031 inline sunrealtype s2c(sunrealtype const *udata, const int dPD) {
00032     return -1.0 / 2.0 * udata[-1 * dPD] + 0 + 1.0 / 2.0 * udata[1 * dPD];
00033 }
00034 #pragma omp declare simd uniform(dPD) notinbranch
00035 inline sunrealtype s2b(sunrealtype const *udata, const int dPD) {
00036     return -3.0 / 2.0 * udata[0] + 2.0 / 1.0 * udata[1 * dPD] -
00037         1.0 / 2.0 * udata[2 * dPD];
00038 }
00039 #pragma omp declare simd uniform(dPD) notinbranch
00040 inline sunrealtype s3f(sunrealtype const *udata, const int dPD) {
00041     return 1.0 / 6.0 * udata[-2 * dPD] - 1.0 / 1.0 * udata[-1 * dPD] +
00042         1.0 / 2.0 * udata[0] + 1.0 / 3.0 * udata[1 * dPD];
00043 }
00044 #pragma omp declare simd uniform(dPD) notinbranch
00045 inline sunrealtype s3b(sunrealtype const *udata, const int dPD) {
00046     return -1.0 / 3.0 * udata[-1 * dPD] - 1.0 / 2.0 * udata[0] + udata[1 * dPD] -
00047         1.0 / 6.0 * udata[2 * dPD];
00048 }
00049 #pragma omp declare simd uniform(dPD) notinbranch
00050 inline sunrealtype s4f(sunrealtype const *udata, const int dPD) {
00051     return -1.0 / 12.0 * udata[-3 * dPD] + 1.0 / 2.0 * udata[-2 * dPD] -
00052         3.0 / 2.0 * udata[-1 * dPD] + 5.0 / 6.0 * udata[0] +
00053         1.0 / 4.0 * udata[1 * dPD];
00054 }
00055 #pragma omp declare simd uniform(dPD) notinbranch
00056 inline sunrealtype s4c(sunrealtype const *udata, const int dPD) {
00057     return 1.0 / 12.0 * udata[-2 * dPD] - 2.0 / 3.0 * udata[-1 * dPD] + 0 +
00058         2.0 / 3.0 * udata[1 * dPD] - 1.0 / 12.0 * udata[2 * dPD];
00059 }
00060 #pragma omp declare simd uniform(dPD) notinbranch
00061 inline sunrealtype s4b(sunrealtype const *udata, const int dPD) {
00062     return -1.0 / 4.0 * udata[-1 * dPD] - 5.0 / 6.0 * udata[0] +
00063         3.0 / 2.0 * udata[1 * dPD] - 1.0 / 2.0 * udata[2 * dPD] +
00064         1.0 / 12.0 * udata[3 * dPD];
00065 }
00066 #pragma omp declare simd uniform(dPD) notinbranch
00067 inline sunrealtype s5f(sunrealtype const *udata, const int dPD) {
00068     return -1.0 / 30.0 * udata[-3 * dPD] + 1.0 / 4.0 * udata[-2 * dPD] -
00069         1.0 / 1.0 * udata[-1 * dPD] + 1.0 / 3.0 * udata[0] +
00070         1.0 / 2.0 * udata[1 * dPD] - 1.0 / 20.0 * udata[2 * dPD];
00071 }
00072 #pragma omp declare simd uniform(dPD) notinbranch
00073 inline sunrealtype s5b(sunrealtype const *udata, const int dPD) {
00074     return 1.0 / 20.0 * udata[-2 * dPD] - 1.0 / 2.0 * udata[-1 * dPD] -
00075         1.0 / 3.0 * udata[0] + udata[1 * dPD] - 1.0 / 4.0 * udata[2 * dPD] +
00076         1.0 / 30.0 * udata[3 * dPD];
00077 }
00078 #pragma omp declare simd uniform(dPD) notinbranch
00079 inline sunrealtype s6f(sunrealtype const *udata, const int dPD) {
00080     return 1.0 / 60.0 * udata[-4 * dPD] - 2.0 / 15.0 * udata[-3 * dPD] +
00081         1.0 / 2.0 * udata[-2 * dPD] - 4.0 / 3.0 * udata[-1 * dPD] +
00082         7.0 / 12.0 * udata[0] + 2.0 / 5.0 * udata[1 * dPD] -
00083         1.0 / 30.0 * udata[2 * dPD];
00084 }
00085 #pragma omp declare simd uniform(dPD) notinbranch
00086 inline sunrealtype s6c(sunrealtype const *udata, const int dPD) {
00087     return -1.0 / 60.0 * udata[-3 * dPD] + 3.0 / 20.0 * udata[-2 * dPD] -
00088         3.0 / 4.0 * udata[-1 * dPD] + 0 + 3.0 / 4.0 * udata[1 * dPD] -
00089         3.0 / 20.0 * udata[2 * dPD] + 1.0 / 60.0 * udata[3 * dPD];
00090 }
00091 #pragma omp declare simd uniform(dPD) notinbranch
00092 inline sunrealtype s6b(sunrealtype const *udata, const int dPD) {
00093     return 1.0 / 30.0 * udata[-2 * dPD] - 2.0 / 5.0 * udata[-1 * dPD] -
00094         7.0 / 12.0 * udata[0] + 4.0 / 3.0 * udata[1 * dPD] -
00095         1.0 / 2.0 * udata[2 * dPD] + 2.0 / 15.0 * udata[3 * dPD] -
00096         1.0 / 60.0 * udata[4 * dPD];
00097 }
00098 #pragma omp declare simd uniform(dPD) notinbranch
00099 inline sunrealtype s7f(sunrealtype const *udata, const int dPD) {
00100     return 1.0 / 140.0 * udata[-4 * dPD] - 1.0 / 15.0 * udata[-3 * dPD] +
00101         3.0 / 10.0 * udata[-2 * dPD] - 1.0 / 1.0 * udata[-1 * dPD] +
00102         1.0 / 4.0 * udata[0] + 3.0 / 5.0 * udata[1 * dPD] -
00103         1.0 / 10.0 * udata[2 * dPD] + 1.0 / 105.0 * udata[3 * dPD];
00104 }
00105 #pragma omp declare simd uniform(dPD) notinbranch
00106 inline sunrealtype s7b(sunrealtype const *udata, const int dPD) {
00107     return -1.0 / 105.0 * udata[-3 * dPD] + 1.0 / 10.0 * udata[-2 * dPD] -
00108         3.0 / 5.0 * udata[-1 * dPD] - 1.0 / 4.0 * udata[0] + udata[1 * dPD] -
00109         3.0 / 10.0 * udata[2 * dPD] + 1.0 / 15.0 * udata[3 * dPD] -
00110         1.0 / 140.0 * udata[4 * dPD];
00111 }
00112 #pragma omp declare simd uniform(dPD) notinbranch
00113 inline sunrealtype s8f(sunrealtype const *udata, const int dPD) {
00114     return -1.0 / 280.0 * udata[-5 * dPD] + 1.0 / 28.0 * udata[-4 * dPD] -

```

```

00115      1.0 / 6.0 * udata[-3 * dPD] + 1.0 / 2.0 * udata[-2 * dPD] -
00116      5.0 / 4.0 * udata[-1 * dPD] + 9.0 / 20.0 * udata[0] +
00117      1.0 / 2.0 * udata[1 * dPD] - 1.0 / 14.0 * udata[2 * dPD] +
00118      1.0 / 168.0 * udata[3 * dPD];
00119 }
00120 #pragma omp declare simd uniform(dPD) notinbranch
00121 inline sunrealtype s8c(sunrealtype const *udata, const int dPD) {
00122     return 1.0 / 280.0 * udata[-4 * dPD] - 4.0 / 105.0 * udata[-3 * dPD] +
00123     1.0 / 5.0 * udata[-2 * dPD] - 4.0 / 5.0 * udata[-1 * dPD] + 0 +
00124     4.0 / 5.0 * udata[1 * dPD] - 1.0 / 5.0 * udata[2 * dPD] +
00125     4.0 / 105.0 * udata[3 * dPD] - 1.0 / 280.0 * udata[4 * dPD];
00126 }
00127 #pragma omp declare simd uniform(dPD) notinbranch
00128 inline sunrealtype s8b(sunrealtype const *udata, const int dPD) {
00129     return -1.0 / 168.0 * udata[-3 * dPD] + 1.0 / 14.0 * udata[-2 * dPD] -
00130     1.0 / 2.0 * udata[-1 * dPD] - 9.0 / 20.0 * udata[0] +
00131     5.0 / 4.0 * udata[1 * dPD] - 1.0 / 2.0 * udata[2 * dPD] +
00132     1.0 / 6.0 * udata[3 * dPD] - 1.0 / 28.0 * udata[4 * dPD] +
00133     1.0 / 280.0 * udata[5 * dPD];
00134 }
00135 #pragma omp declare simd uniform(dPD) notinbranch
00136 inline sunrealtype s9f(sunrealtype const *udata, const int dPD) {
00137     return -1.0 / 630.0 * udata[-5 * dPD] + 1.0 / 56.0 * udata[-4 * dPD] -
00138     2.0 / 21.0 * udata[-3 * dPD] + 1.0 / 3.0 * udata[-2 * dPD] -
00139     1.0 / 1.0 * udata[-1 * dPD] + 1.0 / 5.0 * udata[0] +
00140     2.0 / 3.0 * udata[1 * dPD] - 1.0 / 7.0 * udata[2 * dPD] +
00141     1.0 / 42.0 * udata[3 * dPD] - 1.0 / 504.0 * udata[4 * dPD];
00142 }
00143 #pragma omp declare simd uniform(dPD) notinbranch
00144 inline sunrealtype s9b(sunrealtype const *udata, const int dPD) {
00145     return 1.0 / 504.0 * udata[-4 * dPD] - 1.0 / 42.0 * udata[-3 * dPD] +
00146     1.0 / 7.0 * udata[-2 * dPD] - 2.0 / 3.0 * udata[-1 * dPD] -
00147     1.0 / 5.0 * udata[0] + udata[1 * dPD] - 1.0 / 3.0 * udata[2 * dPD] +
00148     2.0 / 21.0 * udata[3 * dPD] - 1.0 / 56.0 * udata[4 * dPD] +
00149     1.0 / 630.0 * udata[5 * dPD];
00150 }
00151 #pragma omp declare simd uniform(dPD) notinbranch
00152 inline sunrealtype s10f(sunrealtype const *udata, const int dPD) {
00153     return 1.0 / 1260.0 * udata[-6 * dPD] - 1.0 / 105.0 * udata[-5 * dPD] +
00154     3.0 / 56.0 * udata[-4 * dPD] - 4.0 / 21.0 * udata[-3 * dPD] +
00155     1.0 / 2.0 * udata[-2 * dPD] - 6.0 / 5.0 * udata[-1 * dPD] +
00156     11.0 / 30.0 * udata[0] + 4.0 / 7.0 * udata[1 * dPD] -
00157     3.0 / 28.0 * udata[2 * dPD] + 1.0 / 63.0 * udata[3 * dPD] -
00158     1.0 / 840.0 * udata[4 * dPD];
00159 }
00160 #pragma omp declare simd uniform(dPD) notinbranch
00161 inline sunrealtype s10c(sunrealtype const *udata, const int dPD) {
00162     return -1.0 / 1260.0 * udata[-5 * dPD] + 5.0 / 504.0 * udata[-4 * dPD] -
00163     5.0 / 84.0 * udata[-3 * dPD] + 5.0 / 21.0 * udata[-2 * dPD] -
00164     5.0 / 6.0 * udata[-1 * dPD] + 0 + 5.0 / 6.0 * udata[1 * dPD] -
00165     5.0 / 21.0 * udata[2 * dPD] + 5.0 / 84.0 * udata[3 * dPD] -
00166     5.0 / 504.0 * udata[4 * dPD] + 1.0 / 1260.0 * udata[5 * dPD];
00167 }
00168 #pragma omp declare simd uniform(dPD) notinbranch
00169 inline sunrealtype s10b(sunrealtype const *udata, const int dPD) {
00170     return 1.0 / 840.0 * udata[-4 * dPD] - 1.0 / 63.0 * udata[-3 * dPD] +
00171     3.0 / 28.0 * udata[-2 * dPD] - 4.0 / 7.0 * udata[-1 * dPD] -
00172     11.0 / 30.0 * udata[0] + 6.0 / 5.0 * udata[1 * dPD] -
00173     1.0 / 2.0 * udata[2 * dPD] + 4.0 / 21.0 * udata[3 * dPD] -
00174     3.0 / 56.0 * udata[4 * dPD] + 1.0 / 105.0 * udata[5 * dPD] -
00175     1.0 / 1260.0 * udata[6 * dPD];
00176 }
00177 #pragma omp declare simd uniform(dPD) notinbranch
00178 inline sunrealtype s11f(sunrealtype const *udata, const int dPD) {
00179     return 1.0 / 2772.0 * udata[-6 * dPD] - 1.0 / 210.0 * udata[-5 * dPD] +
00180     5.0 / 168.0 * udata[-4 * dPD] - 5.0 / 42.0 * udata[-3 * dPD] +
00181     5.0 / 14.0 * udata[-2 * dPD] - 1.0 / 1.0 * udata[-1 * dPD] +
00182     1.0 / 6.0 * udata[0] + 5.0 / 7.0 * udata[1 * dPD] -
00183     5.0 / 28.0 * udata[2 * dPD] + 5.0 / 126.0 * udata[3 * dPD] -
00184     1.0 / 168.0 * udata[4 * dPD] + 1.0 / 2310.0 * udata[5 * dPD];
00185 }
00186 #pragma omp declare simd uniform(dPD) notinbranch
00187 inline sunrealtype s11b(sunrealtype const *udata, const int dPD) {
00188     return -1.0 / 2310.0 * udata[-5 * dPD] + 1.0 / 168.0 * udata[-4 * dPD] -
00189     5.0 / 126.0 * udata[-3 * dPD] + 5.0 / 28.0 * udata[-2 * dPD] -
00190     5.0 / 7.0 * udata[-1 * dPD] - 1.0 / 6.0 * udata[0] + udata[1 * dPD] -
00191     5.0 / 14.0 * udata[2 * dPD] + 5.0 / 42.0 * udata[3 * dPD] -
00192     5.0 / 168.0 * udata[4 * dPD] + 1.0 / 210.0 * udata[5 * dPD] -
00193     1.0 / 2772.0 * udata[6 * dPD];
00194 }
00195 #pragma omp declare simd uniform(dPD) notinbranch
00196 inline sunrealtype s12f(sunrealtype const *udata, const int dPD) {
00197     return -1.0 / 5544.0 * udata[-7 * dPD] + 1.0 / 396.0 * udata[-6 * dPD] -
00198     1.0 / 60.0 * udata[-5 * dPD] + 5.0 / 72.0 * udata[-4 * dPD] -
00199     5.0 / 24.0 * udata[-3 * dPD] + 1.0 / 2.0 * udata[-2 * dPD] -
00200     7.0 / 6.0 * udata[-1 * dPD] + 13.0 / 42.0 * udata[0] +
00201     5.0 / 8.0 * udata[1 * dPD] - 5.0 / 36.0 * udata[2 * dPD] +

```

```

00202      1.0 / 36.0 * udata[3 * dPD] - 1.0 / 264.0 * udata[4 * dPD] +
00203      1.0 / 3960.0 * udata[5 * dPD];
00204 }
00205 #pragma omp declare simd uniform(dPD) notinbranch
00206 inline sunrealtype s12c(sunrealtype const *udata, const int dPD) {
00207     return 1.0 / 5544.0 * udata[-6 * dPD] - 1.0 / 385.0 * udata[-5 * dPD] +
00208         1.0 / 56.0 * udata[-4 * dPD] - 5.0 / 63.0 * udata[-3 * dPD] +
00209         15.0 / 56.0 * udata[-2 * dPD] - 6.0 / 7.0 * udata[-1 * dPD] + 0 +
00210         6.0 / 7.0 * udata[1 * dPD] - 15.0 / 56.0 * udata[2 * dPD] +
00211         5.0 / 63.0 * udata[3 * dPD] - 1.0 / 56.0 * udata[4 * dPD] +
00212         1.0 / 385.0 * udata[5 * dPD] - 1.0 / 5544.0 * udata[6 * dPD];
00213 }
00214 #pragma omp declare simd uniform(dPD) notinbranch
00215 inline sunrealtype s12b(sunrealtype const *udata, const int dPD) {
00216     return -1.0 / 3960.0 * udata[-5 * dPD] + 1.0 / 264.0 * udata[-4 * dPD] -
00217         1.0 / 36.0 * udata[-3 * dPD] + 5.0 / 36.0 * udata[-2 * dPD] -
00218         5.0 / 8.0 * udata[-1 * dPD] - 13.0 / 42.0 * udata[0] +
00219         7.0 / 6.0 * udata[1 * dPD] - 1.0 / 2.0 * udata[2 * dPD] +
00220         5.0 / 24.0 * udata[3 * dPD] - 5.0 / 72.0 * udata[4 * dPD] +
00221         1.0 / 60.0 * udata[5 * dPD] - 1.0 / 396.0 * udata[6 * dPD] +
00222         1.0 / 5544.0 * udata[7 * dPD];
00223 }
00224 #pragma omp declare simd uniform(dPD) notinbranch
00225 inline sunrealtype s13f(sunrealtype const *udata, const int dPD) {
00226     return -1.0 / 12012.0 * udata[-7 * dPD] + 1.0 / 792.0 * udata[-6 * dPD] -
00227         1.0 / 110.0 * udata[-5 * dPD] + 1.0 / 24.0 * udata[-4 * dPD] -
00228         5.0 / 36.0 * udata[-3 * dPD] + 3.0 / 8.0 * udata[-2 * dPD] -
00229         1.0 / 1.0 * udata[-1 * dPD] + 1.0 / 7.0 * udata[0] +
00230         3.0 / 4.0 * udata[1 * dPD] - 5.0 / 24.0 * udata[2 * dPD] +
00231         1.0 / 18.0 * udata[3 * dPD] - 1.0 / 88.0 * udata[4 * dPD] +
00232         1.0 / 660.0 * udata[5 * dPD] - 1.0 / 10296.0 * udata[6 * dPD];
00233 }
00234 #pragma omp declare simd uniform(dPD) notinbranch
00235 inline sunrealtype s13b(sunrealtype const *udata, const int dPD) {
00236     return 1.0 / 10296.0 * udata[-6 * dPD] - 1.0 / 660.0 * udata[-5 * dPD] +
00237         1.0 / 88.0 * udata[-4 * dPD] - 1.0 / 18.0 * udata[-3 * dPD] +
00238         5.0 / 24.0 * udata[-2 * dPD] - 3.0 / 4.0 * udata[-1 * dPD] -
00239         1.0 / 7.0 * udata[0] + udata[1 * dPD] - 3.0 / 8.0 * udata[2 * dPD] +
00240         5.0 / 36.0 * udata[3 * dPD] - 1.0 / 24.0 * udata[4 * dPD] +
00241         1.0 / 110.0 * udata[5 * dPD] - 1.0 / 792.0 * udata[6 * dPD] +
00242         1.0 / 12012.0 * udata[7 * dPD];
00243 }
00244
00245 /////////////////
00246 // Stencils with dPD fixed to 6 //
00247 /////////////////
00248
00249 inline sunrealtype slf(sunrealtype const *udata) { return slf(udata, 6); }
00250 inline sunrealtype slb(sunrealtype const *udata) { return slb(udata, 6); }
00251 inline sunrealtype s2f(sunrealtype const *udata) { return s2f(udata, 6); }
00252 inline sunrealtype s2c(sunrealtype const *udata) { return s2c(udata, 6); }
00253 inline sunrealtype s2b(sunrealtype const *udata) { return s2b(udata, 6); }
00254 inline sunrealtype s3f(sunrealtype const *udata) { return s3f(udata, 6); }
00255 inline sunrealtype s3b(sunrealtype const *udata) { return s3b(udata, 6); }
00256 inline sunrealtype s4f(sunrealtype const *udata) { return s4f(udata, 6); }
00257 inline sunrealtype s4c(sunrealtype const *udata) { return s4c(udata, 6); }
00258 inline sunrealtype s4b(sunrealtype const *udata) { return s4b(udata, 6); }
00259 inline sunrealtype s5f(sunrealtype const *udata) { return s5f(udata, 6); }
00260 inline sunrealtype s5b(sunrealtype const *udata) { return s5b(udata, 6); }
00261 inline sunrealtype s6f(sunrealtype const *udata) { return s6f(udata, 6); }
00262 inline sunrealtype s6c(sunrealtype const *udata) { return s6c(udata, 6); }
00263 inline sunrealtype s6b(sunrealtype const *udata) { return s6b(udata, 6); }
00264 inline sunrealtype s7f(sunrealtype const *udata) { return s7f(udata, 6); }
00265 inline sunrealtype s7b(sunrealtype const *udata) { return s7b(udata, 6); }
00266 inline sunrealtype s8f(sunrealtype const *udata) { return s8f(udata, 6); }
00267 inline sunrealtype s8c(sunrealtype const *udata) { return s8c(udata, 6); }
00268 inline sunrealtype s8b(sunrealtype const *udata) { return s8b(udata, 6); }
00269 inline sunrealtype s9f(sunrealtype const *udata) { return s9f(udata, 6); }
00270 inline sunrealtype s9b(sunrealtype const *udata) { return s9b(udata, 6); }
00271 inline sunrealtype s10f(sunrealtype const *udata)
00272 { return s10f(udata, 6); }
00273 inline sunrealtype s10c(sunrealtype const *udata)
00274 { return s10c(udata, 6); }
00275 inline sunrealtype s10b(sunrealtype const *udata)
00276 { return s10b(udata, 6); }
00277 inline sunrealtype s11f(sunrealtype const *udata)
00278 { return s11f(udata, 6); }
00279 inline sunrealtype s11b(sunrealtype const *udata)
00280 { return s11b(udata, 6); }
00281 inline sunrealtype s12f(sunrealtype const *udata)
00282 { return s12f(udata, 6); }
00283 inline sunrealtype s12c(sunrealtype const *udata)
00284 { return s12c(udata, 6); }
00285 inline sunrealtype s12b(sunrealtype const *udata)
00286 { return s12b(udata, 6); }
00287 inline sunrealtype s13f(sunrealtype const *udata)
00288 { return s13f(udata, 6); }

```

```

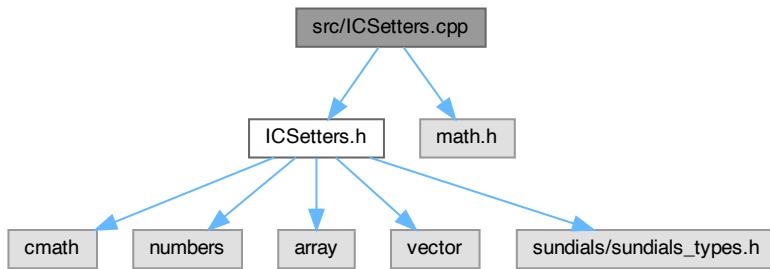
00289 inline sunrealtype s13b(sunrealtype const *udata)
00290 { return s13b(udata, 6); }
00291

```

## 6.6 src/ICSetters.cpp File Reference

Implementation of the plane wave and Gaussian wave packets.

```
#include "ICSetters.h"
#include <math.h>
Include dependency graph for ICSetters.cpp:
```



### 6.6.1 Detailed Description

Implementation of the plane wave and Gaussian wave packets.

Definition in file [ICSetters.cpp](#).

## 6.7 ICSetters.cpp

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file ICSetters.cpp
00003 /// @brief Implementation of the plane wave and Gaussian wave packets
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #include "ICSetters.h"
00007
00008 #include <math.h>
00009
00010 /** PlaneWave1D construction with */
00011 PlaneWave1D::PlaneWave1D(std::array<sunrealtype, 3> k,
00012     std::array<sunrealtype, 3> p,
00013     std::array<sunrealtype, 3> phi) {
00014     kx = k[0]; /** - wavevectors \f$ k_x \f$ */
00015     ky = k[1]; /** - \f$ k_y \f$ */
00016     kz = k[2]; /** - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00017     // Amplitude bug: lower by factor 3
00018     px = p[0] / 3; /** - amplitude (polarization) in x-direction \f$ p_x \f$ */
00019     py = p[1] / 3; /** - amplitude (polarization) in y-direction \f$ p_y \f$ */
00020     pz = p[2] / 3; /** - amplitude (polarization) in z-direction \f$ p_z \f$ */
00021     phix = phi[0]; /** - phase shift in x-direction \f$ \phi_x \f$ */
00022     phiy = phi[1]; /** - phase shift in y-direction \f$ \phi_y \f$ */
00023     phiz = phi[2]; /** - phase shift in z-direction \f$ \phi_z \f$ */

```

```

00024 }
00025
00026 /** PlaneWave1D implementation in space */
00027 void PlaneWave1D::addToSpace(const sunrealtype x, const sunrealtype y,
00028     const sunrealtype z,
00029     sunrealtype *pTo6Space) const {
00030     const sunrealtype wavelength =
00031         sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00032     const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00033         std::numbers::pi; /* \f$ 2\pi \ \vec{k} \cdot \vec{E} \f$ */
00034     // Plane wave definition
00035     const std::array<sunrealtype, 3> E{{
00036         px * cos(kScalarX - phix), /* \f$ E_x \f$ */
00037         py * cos(kScalarX - phiy), /* \f$ E_y \f$ */
00038         pz * cos(kScalarX - phiz)} }; /* \f$ E_z \f$ */
00039     // Put E-field into space
00040     pTo6Space[0] += E[0];
00041     pTo6Space[1] += E[1];
00042     pTo6Space[2] += E[2];
00043     // and B-field
00044     pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00045     pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00046     pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00047 }
00048
00049 /** PlaneWave2D construction with */
00050 PlaneWave2D::PlaneWave2D(std::array<sunrealtype, 3> k,
00051     std::array<sunrealtype, 3> p,
00052     std::array<sunrealtype, 3> phi) {
00053     kx = k[0]; /* - wavevectors \f$ k_x \f$ */
00054     ky = k[1]; /* - \f$ k_y \f$ */
00055     kz = k[2]; /* - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00056     // Amplitude bug: lower by factor 9
00057     px = p[0] / 9; /* - amplitude (polarization) in x-direction \f$ p_x \f$ */
00058     py = p[1] / 9; /* - amplitude (polarization) in y-direction \f$ p_y \f$ */
00059     pz = p[2] / 9; /* - amplitude (polarization) in z-direction \f$ p_z \f$ */
00060     phix = phi[0]; /* - phase shift in x-direction \f$ \phi_x \f$ */
00061     phiy = phi[1]; /* - phase shift in y-direction \f$ \phi_y \f$ */
00062     phiz = phi[2]; /* - phase shift in z-direction \f$ \phi_z \f$ */
00063 }
00064
00065 /** PlaneWave2D implementation in space */
00066 void PlaneWave2D::addToSpace(const sunrealtype x, const sunrealtype y,
00067     const sunrealtype z, sunrealtype *pTo6Space) const {
00068     const sunrealtype wavelength =
00069         sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00070     const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00071         std::numbers::pi; /* \f$ 2\pi \ \vec{k} \cdot \vec{E} \f$ */
00072     // Plane wave definition
00073     const std::array<sunrealtype, 3> E{{
00074         px * cos(kScalarX - phix), /* \f$ E_x \f$ */
00075         py * cos(kScalarX - phiy), /* \f$ E_y \f$ */
00076         pz * cos(kScalarX - phiz)} }; /* \f$ E_z \f$ */
00077     // Put E-field into space
00078     pTo6Space[0] += E[0];
00079     pTo6Space[1] += E[1];
00080     pTo6Space[2] += E[2];
00081     // and B-field
00082     pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00083     pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00084     pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00085 }
00086
00087 /** PlaneWave3D construction with */
00088 PlaneWave3D::PlaneWave3D(std::array<sunrealtype, 3> k,
00089     std::array<sunrealtype, 3> p,
00090     std::array<sunrealtype, 3> phi) {
00091     kx = k[0]; /* - wavevectors \f$ k_x \f$ */
00092     ky = k[1]; /* - \f$ k_y \f$ */
00093     kz = k[2]; /* - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00094     px = p[0]; /* - amplitude (polarization) in x-direction \f$ p_x \f$ */
00095     py = p[1]; /* - amplitude (polarization) in y-direction \f$ p_y \f$ */
00096     pz = p[2]; /* - amplitude (polarization) in z-direction \f$ p_z \f$ */
00097     phix = phi[0]; /* - phase shift in x-direction \f$ \phi_x \f$ */
00098     phiy = phi[1]; /* - phase shift in y-direction \f$ \phi_y \f$ */
00099     phiz = phi[2]; /* - phase shift in z-direction \f$ \phi_z \f$ */
00100 }
00101
00102 /** PlaneWave3D implementation in space */
00103 void PlaneWave3D::addToSpace(sunrealtype x, sunrealtype y, sunrealtype z,
00104     sunrealtype *pTo6Space) const {
00105     const sunrealtype wavelength =
00106         sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00107     const sunrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00108         std::numbers::pi; /* \f$ 2\pi \ \vec{k} \cdot \vec{E} \f$ */
00109     // Plane wave definition
00110     const std::array<sunrealtype, 3> E{/* E-field vector \f$ \vec{E} \f$ */

```

```

00111           px * cos(kScalarX - phix), /* \f$ E_x \f$ */
00112           py * cos(kScalarX - phiy), /* \f$ E_y \f$ */
00113           pz * cos(kScalarX - phiz)}); /* \f$ E_z \f$ */
00114 // Put E-field into space
00115 pTo6Space[0] += E[0];
00116 pTo6Space[1] += E[1];
00117 pTo6Space[2] += E[2];
00118 // and B-field
00119 pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00120 pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00121 pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00122 }
00123
00124 /** Gauss1D construction with */
00125 Gauss1D::Gauss1D(std::array<sunrealtype, 3> k, std::array<sunrealtype, 3> p,
00126                     std::array<sunrealtype, 3> xo, unrealtype phig_,
00127                     std::array<sunrealtype, 3> phi) {
00128   kx = k[0]; /* - wavevectors \f$ k_x \f$ */
00129   ky = k[1]; /* - \f$ k_y \f$ */
00130   kz = k[2]; /* - \f$ k_z \f$ normalized to \f$ 1/\lambda \f$ */
00131   px = p[0]; /* - amplitude (polarization) in x-direction */
00132   py = p[1]; /* - amplitude (polarization) in y-direction */
00133   pz = p[2]; /* - amplitude (polarization) in z-direction */
00134   phix = phi[0]; /* - phase shift in x-direction */
00135   phiy = phi[1]; /* - phase shift in y-direction */
00136   phiz = phi[2]; /* - phase shift in z-direction */
00137   phig = phig_; /* - width */
00138   x0x = xo[0]; /* - shift from origin in x-direction*/
00139   x0y = xo[1]; /* - shift from origin in y-direction*/
00140   x0z = xo[2]; /* - shift from origin in z-direction*/
00141 }
00142
00143 /** Gauss1D implementation in space */
00144 void Gauss1D::addToSpace(unrealtype x, unrealtype y, unrealtype z,
00145                           unrealtype *pTo6Space) const {
00146   const unrealtype wavelength =
00147     sqrt(kx * kx + ky * ky + kz * kz); /* \f$ 1/\lambda \f$ */
00148   x = x - x0x; /* x-coordinate minus shift from origin */
00149   y = y - x0y; /* y-coordinate minus shift from origin */
00150   z = z - x0z; /* z-coordinate minus shift from origin */
00151   const unrealtype kScalarX = (kx * x + ky * y + kz * z) * 2 *
00152     std::numbers::pi; /* \f$ 2\pi \f$ */
00153   const unrealtype envelopeAmp =
00154     exp(-(x * x + y * y + z * z) / phig / phig); /* enveloping Gauss shape */
00155 // Gaussian wave definition
00156   const std::array<sunrealtype, 3> E{
00157     {px * cos(kScalarX - phix) * envelopeAmp, /* \f$ E_x \f$ */
00158      py * cos(kScalarX - phiy) * envelopeAmp, /* \f$ E_y \f$ */
00159      pz * cos(kScalarX - phiz) * envelopeAmp}; /* \f$ E_z \f$ */
00160 // Put E-field into space
00161   pTo6Space[0] += E[0];
00162   pTo6Space[1] += E[1];
00163   pTo6Space[2] += E[2];
00164 // and B-field
00165   pTo6Space[3] += (ky * E[2] - kz * E[1]) / wavelength;
00166   pTo6Space[4] += (kz * E[0] - kx * E[2]) / wavelength;
00167   pTo6Space[5] += (kx * E[1] - ky * E[0]) / wavelength;
00168 }
00169
00170
00171 /** Gauss2D construction with */
00172 Gauss2D::Gauss2D(std::array<sunrealtype, 3> dis_,
00173                     std::array<sunrealtype, 3> axis_,
00174                     unrealtype Amp_, unrealtype phip_, unrealtype w0_,
00175                     unrealtype zr_, unrealtype Ph0_, unrealtype PhA_) {
00176   dis = dis_; /* - center it approaches */
00177   axis = axis_; /* - direction form where it comes */
00178   Amp = Amp_; /* - amplitude */
00179   phip = phip_; /* - polarization rotation from TE-mode */
00180   w0 = w0_; /* - taille */
00181   zr = zr_; /* - Rayleigh length */
00182   Ph0 = Ph0_; /* - beam center */
00183   PhA = PhA_; /* - beam length */
00184   A1 = Amp * cos(phip); /* amplitude in z-direction */
00185   A2 = Amp * sin(phip); /* amplitude on xy-plane */
00186   lambda = std::numbers::pi * w0 * w0 / zr; /* formula for wavelength */
00187 }
00188
00189 void Gauss2D::addToSpace(unrealtype x, unrealtype y, unrealtype z,
00190                           unrealtype *pTo6Space) const {
00191   // \f$ \vec{x} = \vec{x}_0 - \vec{dis} \f$ // coordinates minus distance to
00192   // origin
00193   x -= dis[0];
00194   y -= dis[1];
00195   // z-=dis[2];
00196   z = nan("0x12345"); /* unused parameter */
00197   // \f$ z_g = \vec{x} \cdot \vec{e}_g \f$ projection on propagation axis

```

```

00198 const sunrealtypet zg =
00199     x * axis[0] + y * axis[1]; //+z*axis[2]; // =z-z0 -> propagation
00200                                     //direction, minus origin
00201 // \f$ r = \sqrt{\vec{x}^2 - z_g^2} \f$ -> pythagoras of radius minus
00202 // projection on prop axis
00203 const sunrealtypet r = sqrt((x * x + y * y /*+z*z*/) -
00204     zg * zg); // radial distance to propagation axis
00205 // \f$ w(z) = w_0\sqrt{1+(z_g/z_R)^2} \f$
00206 // waist at position z
00207 const sunrealtypet wz = w0 * sqrt(1 + (zg * zg / zr / zr));
00208 // \f$ g(z) = atan(z_g/z_r) \f$
00209 const sunrealtypet gz = atan(zg / zr); // Gouy phase
00210 // \f$ R(z) = z_g*(1+(z_r/z_g)^2) \f$
00211 sunrealtypet Rz = nan("0x12345"); // beam curvature
00212 if (abs(zg) > 1e-15)
00213     Rz = zg * (1 + (zr * zr / zg / zg));
00214 else
00215     Rz = 1e308;
00216 // wavenumber \f$ k = 2\pi/\lambda \f$
00217 const sunrealtypet k = 2 * std::numbers::pi / lambda;
00218 // \f$ \Phi_F = kr^2/(2*R(z))+g(z)-kz_g \f$
00219 const sunrealtypet PhF =
00220     -k * r * r / (2 * Rz) + gz - k * zg; // to be inserted into cosine
00221 // \f$ G = \sqrt{w_0/w_z}\mathrm{e}^{-(r/w_z)^2}\mathrm{e}^{-(z-g-Ph0)^2/PhA^2}\cos(\Phi_F) \f$
00222 // CVode is a diva, no chance to remove the square in the second exponential
00223 // -> h too small
00224 const sunrealtypet G2D = sqrt(w0 / wz) * exp(-r * r / wz / wz) *
00225     exp(-(zg - Ph0) * (zg - Ph0) / PhA / PhA) *
00226     cos(PhF); // gauss shape
00227 // \f$ c_\alpha = \vec{e}_x \cdot \vec{c} \cdot \vec{a} \f$
00228 // projection components; do like this for CVode convergence -> otherwise
00229 // results in machine error values for non-existant field components if
00230 // axis[0] and axis[1] are given
00231 const sunrealtypet ca =
00232     axis[0]; // x-component of propagation axis which is given as parameter
00233 // no z-component for 2D propagation
00234 const sunrealtypet sa = sqrt(1 - ca * ca);
00235 // E-field to space: polarization in xy-plane (A2) is projection of
00236 // z-polarization (A1) on x- and y-directions
00237 pTo6Space[0] += sa * (G2D * A2);
00238 pTo6Space[1] += -ca * (G2D * A2);
00239 pTo6Space[2] += G2D * A1;
00240 // B-field -> negative derivative wrt polarization shift of E-field
00241 pTo6Space[3] += -sa * (G2D * A1);
00242 pTo6Space[4] += ca * (G2D * A1);
00243 pTo6Space[5] += G2D * A2;
00244 }
00245
00246 /** Gauss3D construction with */
00247 Gauss3D::Gauss3D(std::array<sunrealtypet, 3> dis_,
00248                     std::array<sunrealtypet, 3> axis_,
00249                     sunrealtypet Amp_,
00250                     // std::array<sunrealtypet, 3> pol_,
00251                     sunrealtypet phip_, sunrealtypet w0_, sunrealtypet zr_,
00252                     sunrealtypet Ph0_, sunrealtypet PhA_) {
00253     dis = dis_; //** - center it approaches */
00254     axis = axis_; //** - direction from where it comes */
00255     Amp = Amp_; //** - amplitude */
00256     // pol=pol_;
00257     phip = phip_; //** - polarization rotation form TE-mode */
00258     w0 = w0_; //** - taille */
00259     zr = zr_; //** - Rayleigh length */
00260     Ph0 = Ph0_; //** - beam center */
00261     PhA = PhA_; //** - beam length */
00262     lambda = std::numbers::pi * w0 * w0 / zr;
00263     A1 = Amp * cos(phip);
00264     A2 = Amp * sin(phip);
00265 }
00266
00267 /** Gauss3D implementation in space */
00268 void Gauss3D::addToSpace(sunrealtypet x, sunrealtypet y, sunrealtypet z,
00269                           sunrealtypet *pTo6Space) const {
00270     x -= dis[0];
00271     y -= dis[1];
00272     z -= dis[2];
00273     const sunrealtypet zg = x * axis[0] + y * axis[1] + z * axis[2];
00274     const sunrealtypet r = sqrt((x * x + y * y + z * z) - zg * zg);
00275     const sunrealtypet wz = w0 * sqrt(1 + (zg * zg / zr / zr));
00276     const sunrealtypet gz = atan(zg / zr);
00277     sunrealtypet Rz = nan("0x12345");
00278     if (abs(zg) > 1e-15)
00279         Rz = zg * (1 + (zr * zr / zg / zg));
00280     else
00281         Rz = 1e308;
00282     const sunrealtypet k = 2 * std::numbers::pi / lambda;
00283     const sunrealtypet PhF = -k * r * r / (2 * Rz) + gz - k * zg;
00284     const sunrealtypet G3D = (w0 / wz) * exp(-r * r / wz / wz) *

```

```

00285           exp(-(zg - Ph0) * (zg - Ph0) / PhA / PhA) * cos(PhF);
00286   const sunrealtypes ca = axis[0];
00287   const sunrealtypes sa = sqrt(1 - ca * ca);
00288   pTo6Space[0] += sa * (G3D * A2);
00289   pTo6Space[1] += -ca * (G3D * A2);
00290   pTo6Space[2] += G3D * A1;
00291   pTo6Space[3] += -sa * (G3D * A1);
00292   pTo6Space[4] += ca * (G3D * A1);
00293   pTo6Space[5] += G3D * A2;
00294 }
00295
00296 /** Evaluate lattice point values to zero and then add initial field values */
00297 void ICSetter::eval(sunrealtypes x, sunrealtypes y, sunrealtypes z,
00298                      sunrealtypes *pTo6Space) {
00299   pTo6Space[0] = 0;
00300   pTo6Space[1] = 0;
00301   pTo6Space[2] = 0;
00302   pTo6Space[3] = 0;
00303   pTo6Space[4] = 0;
00304   pTo6Space[5] = 0;
00305   add(x, y, z, pTo6Space);
00306 }
00307
00308 /** Add all initial field values to the lattice space */
00309 void ICSetter::add(sunrealtypes x, sunrealtypes y, sunrealtypes z,
00310                      sunrealtypes *pTo6Space) {
00311   for (const auto &wave : planeWaves1D)
00312     wave.addToSpace(x, y, z, pTo6Space);
00313   for (const auto &wave : planeWaves2D)
00314     wave.addToSpace(x, y, z, pTo6Space);
00315   for (const auto &wave : planeWaves3D)
00316     wave.addToSpace(x, y, z, pTo6Space);
00317   for (const auto &wave : gauss1Ds)
00318     wave.addToSpace(x, y, z, pTo6Space);
00319   for (const auto &wave : gauss2Ds)
00320     wave.addToSpace(x, y, z, pTo6Space);
00321   for (const auto &wave : gauss3Ds)
00322     wave.addToSpace(x, y, z, pTo6Space);
00323 }
00324
00325 /** Add plane waves in 1D to their container vector */
00326 void ICSetter::addPlaneWave1D(std::array<sunrealtypes, 3> k,
00327                               std::array<sunrealtypes, 3> p,
00328                               std::array<sunrealtypes, 3> phi) {
00329   planeWaves1D.emplace_back(PlaneWave1D(k, p, phi));
00330 }
00331
00332 /** Add plane waves in 2D to their container vector */
00333 void ICSetter::addPlaneWave2D(std::array<sunrealtypes, 3> k,
00334                               std::array<sunrealtypes, 3> p,
00335                               std::array<sunrealtypes, 3> phi) {
00336   planeWaves2D.emplace_back(PlaneWave2D(k, p, phi));
00337 }
00338
00339 /** Add plane waves in 3D to their container vector */
00340 void ICSetter::addPlaneWave3D(std::array<sunrealtypes, 3> k,
00341                               std::array<sunrealtypes, 3> p,
00342                               std::array<sunrealtypes, 3> phi) {
00343   planeWaves3D.emplace_back(PlaneWave3D(k, p, phi));
00344 }
00345
00346 /** Add Gaussian waves in 1D to their container vector */
00347 void ICSetter::addGauss1D(std::array<sunrealtypes, 3> k,
00348                           std::array<sunrealtypes, 3> p,
00349                           std::array<sunrealtypes, 3> xo, sunrealtypes phig_,
00350                           std::array<sunrealtypes, 3> phi) {
00351   gauss1Ds.emplace_back(Gauss1D(k, p, xo, phig_, phi));
00352 }
00353
00354 /** Add Gaussian waves in 2D to their container vector */
00355 void ICSetter::addGauss2D(std::array<sunrealtypes, 3> dis_,
00356                           std::array<sunrealtypes, 3> axis_,
00357                           sunrealtypes Amp_, sunrealtypes phip_, sunrealtypes w0_,
00358                           sunrealtypes zr_, sunrealtypes Ph0_, sunrealtypes PhA_)
00359 {
00360   gauss2Ds.emplace_back(
00361     Gauss2D(dis_, axis_, Amp_, phip_, w0_, zr_, Ph0_, PhA_));
00362 }
00363
00364 /** Add Gaussian waves in 3D to their container vector */
00365 void ICSetter::addGauss3D(std::array<sunrealtypes, 3> dis_,
00366                           std::array<sunrealtypes, 3> axis_,
00367                           sunrealtypes Amp_, sunrealtypes phip_, sunrealtypes w0_,
00368                           sunrealtypes zr_, sunrealtypes Ph0_, sunrealtypes PhA_)
00369 {
00370   gauss3Ds.emplace_back(
00371     Gauss3D(dis_, axis_, Amp_, phip_, w0_, zr_, Ph0_, PhA_));

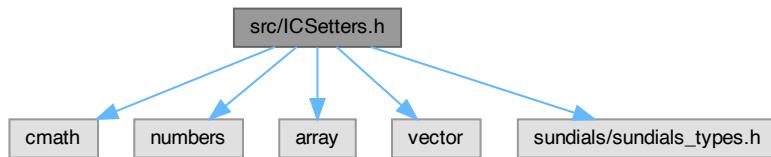
```

```
00372 }
```

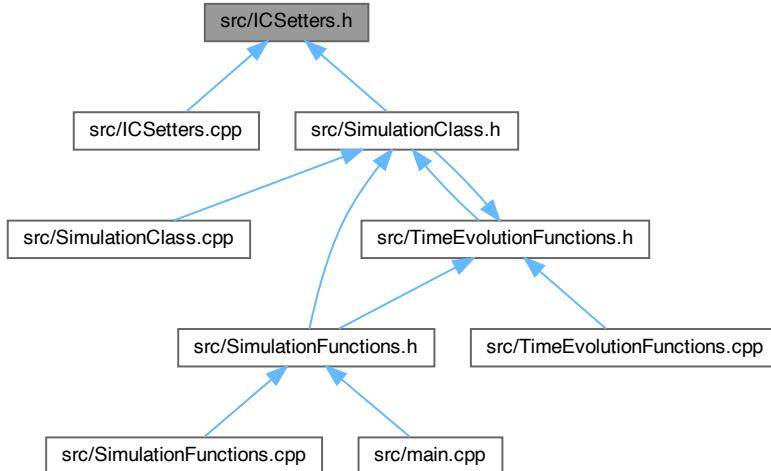
## 6.8 src/ICSetters.h File Reference

Declaration of the plane wave and Gaussian wave packets.

```
#include <cmath>
#include <numbers>
#include <array>
#include <vector>
#include <sundials/sundials_types.h>
Include dependency graph for ICSetters.h:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [PlaneWave](#)  
*super-class for plane waves*
- class [PlaneWave1D](#)

*class for plane waves in 1D*

- class [PlaneWave2D](#)  
*class for plane waves in 2D*
- class [PlaneWave3D](#)  
*class for plane waves in 3D*
- class [Gauss1D](#)  
*class for Gaussian pulses in 1D*
- class [Gauss2D](#)  
*class for Gaussian pulses in 2D*
- class [Gauss3D](#)  
*class for Gaussian pulses in 3D*
- class [ICSetter](#)

*ICSetter class to initialize wave types with default parameters.*

### 6.8.1 Detailed Description

Declaration of the plane wave and Gaussian wave packets.

Definition in file [ICSetters.h](#).

## 6.9 ICSetters.h

[Go to the documentation of this file.](#)

```
00001 //////////////////////////////////////////////////////////////////
00002 /// @file ICSetters.h
00003 /// @brief Declaration of the plane wave and Gaussian wave packets
00004 //////////////////////////////////////////////////////////////////
00005
00006 #pragma once
00007
00008 // math, constants, vector, and array
00009 #include <cmath>
00010 #include <numbers>
00011 #include <array>
00012 #include <vector>
00013
00014 #include <sundials/sundials_types.h> /* definition of type sunrealtypes */
00015
00016 /** @brief super-class for plane waves
00017 *
00018 * They are given in the form  $\vec{E} = \vec{E}_0 \cos(\vec{k} \cdot \vec{x} - \phi)$ 
00019 */
00020 class PlaneWave {
00021 protected:
00022     /// wavenumber  $k_x$ 
00023     sunrealtypes kx;
00024     /// wavenumber  $k_y$ 
00025     sunrealtypes ky;
00026     /// wavenumber  $k_z$ 
00027     sunrealtypes kz;
00028     /// polarization & amplitude in x-direction,  $p_x$ 
00029     sunrealtypes px;
00030     /// polarization & amplitude in y-direction,  $p_y$ 
00031     sunrealtypes py;
00032     /// polarization & amplitude in z-direction,  $p_z$ 
00033     sunrealtypes pz;
00034     /// phase shift in x-direction,  $\phi_x$ 
00035     sunrealtypes phix;
00036     /// phase shift in y-direction,  $\phi_y$ 
00037     sunrealtypes phiy;
00038     /// phase shift in z-direction,  $\phi_z$ 
00039     sunrealtypes phiz;
00040 };
00041
00042 /** @brief class for plane waves in 1D */
00043 class PlaneWave1D : public PlaneWave {
00044 public:
```

```

00045  /// construction with default parameters
00046  PlaneWave1D(std::array<sunrealtype, 3> k = {1, 0, 0},
00047      std::array<sunrealtype, 3> p = {0, 0, 1},
00048      std::array<sunrealtype, 3> phi = {0, 0, 0});
00049  /// function for the actual implementation in the lattice
00050  void addToSpace(sunrealtype x, unrealtype y, unrealtype z,
00051      unrealtype *pTo6Space) const;
00052 };
00053
00054 /** @brief class for plane waves in 2D */
00055 class PlaneWave2D : public PlaneWave {
00056 public:
00057     /// construction with default parameters
00058     PlaneWave2D(std::array<sunrealtype, 3> k = {1, 0, 0},
00059         std::array<sunrealtype, 3> p = {0, 0, 1},
00060         std::array<sunrealtype, 3> phi = {0, 0, 0});
00061     /// function for the actual implementation in the lattice
00062     void addToSpace(sunrealtype x, unrealtype y, unrealtype z,
00063         unrealtype *pTo6Space) const;
00064 };
00065
00066 /** @brief class for plane waves in 3D */
00067 class PlaneWave3D : public PlaneWave {
00068 public:
00069     /// construction with default parameters
00070     PlaneWave3D(std::array<sunrealtype, 3> k = {1, 0, 0},
00071         std::array<sunrealtype, 3> p = {0, 0, 1},
00072         std::array<sunrealtype, 3> phi = {0, 0, 0});
00073     /// function for the actual implementation in space
00074     void addToSpace(unrealtype x, unrealtype y, unrealtype z,
00075         unrealtype *pTo6Space) const;
00076 };
00077
00078 /** @brief class for Gaussian pulses in 1D
00079 */
00080 * They are given in the form  $\vec{E} = \vec{p} \cdot \exp \left( -(\vec{x} - \vec{x}_0)^2 / \Phi_g^2 \right) \cos(\vec{k} \cdot \vec{x})$ 
00081 */
00082
00083 class Gauss1D {
00084 private:
00085     /// wavenumber  $k_x$ 
00086     unrealtype kx;
00087     /// wavenumber  $k_y$ 
00088     unrealtype ky;
00089     /// wavenumber  $k_z$ 
00090     unrealtype kz;
00091     /// polarization & amplitude in x-direction,  $p_x$ 
00092     unrealtype px;
00093     /// polarization & amplitude in y-direction,  $p_y$ 
00094     unrealtype py;
00095     /// polarization & amplitude in z-direction,  $p_z$ 
00096     unrealtype pz;
00097     /// phase shift in x-direction,  $\phi_x$ 
00098     unrealtype phix;
00099     /// phase shift in y-direction,  $\phi_y$ 
00100    unrealtype phiy;
00101    /// phase shift in z-direction,  $\phi_z$ 
00102    unrealtype phiz;
00103    /// center of pulse in x-direction,  $x_0$ 
00104    unrealtype x0;
00105    /// center of pulse in y-direction,  $y_0$ 
00106    unrealtype y0;
00107    /// center of pulse in z-direction,  $z_0$ 
00108    unrealtype z0;
00109    /// pulse width  $\Phi_g$ 
00110    unrealtype phig;
00111
00112 public:
00113     /// construction with default parameters
00114     Gauss1D(std::array<sunrealtype, 3> k = {1, 0, 0},
00115         std::array<sunrealtype, 3> p = {0, 0, 1},
00116         std::array<sunrealtype, 3> xo = {0, 0, 0},
00117         unrealtype phig_ = 1.0,
00118         std::array<sunrealtype, 3> phi = {0, 0, 0});
00119     /// function for the actual implementation in space
00120     void addToSpace(unrealtype x, unrealtype y, unrealtype z,
00121         unrealtype *pTo6Space) const;
00122
00123 public:
00124 };
00125
00126 /** @brief class for Gaussian pulses in 2D
00127 */
00128 * They are given in the form
00129 *  $\vec{E} = \vec{\epsilon} \cdot \vec{A} \cdot \sqrt{\frac{\omega_0}{\omega(z)}} \exp \left( -r^2 / \Phi_g^2 \right) \exp \left( -((z - z_0) / \Phi_g)^2 \right) \cos \left( \frac{k}{r^2} (2R(z) + g(z) - k) \right)$ 
00130 * with
00131 */

```

```

00132 * - propagation direction (subtracted distance to origin) \f$ z_g \f$
00133 * - radial distance to propagation axis \f$ r = \sqrt{\vec{x}^2 - z_g^2} \f$
00134 * - \f$ k = 2\pi / \lambda \f$
00135 * - waist at position z, \f$ \omega(z) = w_0 \ , \ \sqrt{1+(z_g/z_R)^2} \f$
00136 * - Gouy phase \f$ g(z) = \tan^{-1}(z_g/z_r) \f$
00137 * - beam curvature \f$ R(z) = z_g \ , \ (1+(z_r/z_g)^2) \f$
00138 * obtained via the chosen parameters */
00139 class Gauss2D {
00140 private:
00141     /// distance maximum to origin
00142     std::array<surrealtype, 3> dis;
00143     /// normalized propagation axis
00144     std::array<surrealtype, 3> axis;
00145     /// amplitude \f$ A\f$
00146     surrealtype Amp;
00147     /// polarization rotation from TE-mode around propagation direction
00148     // that determines \f$ \vec{\epsilon}\f$ above
00149     surrealtype phip;
00150     /// taille \f$ \omega_0 \f$
00151     surrealtype w0;
00152     /// Rayleigh length \f$ z_R = \pi \omega_0^2 / \lambda \f$
00153     surrealtype zr;
00154     /// center of beam \f$ \Phi_0 \f$
00155     surrealtype Ph0;
00156     /// length of beam \f$ \Phi_A \f$
00157     surrealtype PhA;
00158     /// amplitude projection on TE-mode
00159     surrealtype A1;
00160     /// amplitude projection on xy-plane
00161     surrealtype A2;
00162     /// wavelength \f$ \lambda \f$
00163     surrealtype lambda;
00164
00165 public:
00166     /// construction with default parameters
00167     Gauss2D(std::array<surrealtype, 3> dis_ = {0, 0, 0},
00168               std::array<surrealtype, 3> axis_ = {1, 0, 0},
00169               surrealtype Amp_ = 1.0,
00170               surrealtype phip_ = 0, surrealtype w0_ = 1e-5,
00171               surrealtype zr_ = 4e-5,
00172               surrealtype Ph0_ = 2e-5, surrealtype PhA_ = 0.45e-5);
00173     /// function for the actual implementation in space
00174     void addToSpace(surrealtype x, surrealtype y, surrealtype z,
00175                     surrealtype *pTo6Space) const;
00176
00177 public:
00178 };
00179
00180 /** @brief class for Gaussian pulses in 3D
00181 *
00182 * They are given in the form
00183 * \f$ \vec{E} = \vec{\epsilon} \exp(-r/\omega(z))^2 \exp(-((z_g-\Phi_0)/\Phi_A)^2) \cos(\frac{k}{r^2}(2R(z)) + g(z) - k) \f$ with
00184 * - propagation direction (subtracted distance to origin) \f$ z_g \f$
00185 * - radial distance to propagation axis \f$ r = \sqrt{\vec{x}^2 - z_g^2} \f$
00186 * - \f$ k = 2\pi / \lambda \f$
00187 * - waist at position z, \f$ \omega(z) = w_0 \ , \ \sqrt{1+(z_g/z_R)^2} \f$
00188 * - Gouy phase \f$ g(z) = \tan^{-1}(z_g/z_r) \f$
00189 * - beam curvature \f$ R(z) = z_g \ , \ (1+(z_r/z_g)^2) \f$
00190 * obtained via the chosen parameters */
00191
00192 class Gauss3D {
00193 private:
00194     /// distance maximum to origin
00195     std::array<surrealtype, 3> dis;
00196     /// normalized propagation axis
00197     std::array<surrealtype, 3> axis;
00198     /// amplitude \f$ A\f$
00199     surrealtype Amp;
00200     /// polarization rotation from TE-mode around propagation direction
00201     // that determines \f$ \vec{\epsilon}\f$ above
00202     surrealtype phip;
00203     // polarization
00204     std::array<surrealtype, 3> pol;
00205     /// taille \f$ \omega_0 \f$
00206     surrealtype w0;
00207     /// Rayleigh length \f$ z_R = \pi \omega_0^2 / \lambda \f$
00208     surrealtype zr;
00209     /// center of beam \f$ \Phi_0 \f$
00210     surrealtype Ph0;
00211     /// length of beam \f$ \Phi_A \f$
00212     surrealtype PhA;
00213     /// amplitude projection on TE-mode (z-axis)
00214     surrealtype A1;
00215     /// amplitude projection on xy-plane
00216     surrealtype A2;
00217     /// wavelength \f$ \lambda \f$
00218

```

```

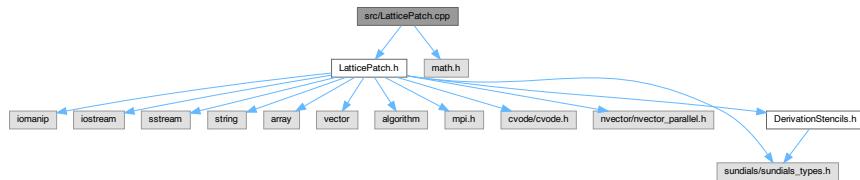
00219     sunrealtype lambda;
00220
00221 public:
00222     /// construction with default parameters
00223     Gauss3D(std::array<sunrealtype, 3> dis_ = {0, 0, 0},
00224             std::array<sunrealtype, 3> axis_ = {1, 0, 0},
00225             unrealtype Amp_ = 1.0,
00226             unrealtype phip_ = 0,
00227             // unrealtype pol_ = {0, 0, 1},
00228             unrealtype w0_ = 1e-5, unrealtype zr_ = 4e-5,
00229             unrealtype Ph0_ = 2e-5, unrealtype PhA_ = 0.45e-5);
00230     /// function for the actual implementation in space
00231     void addToSpace(unrealtype x, unrealtype y, unrealtype z,
00232                     unrealtype *pTo6Space) const;
00233
00234 public:
00235 };
00236
00237 /** @brief ICSsetter class to initialize wave types with default parameters */
00238 class ICSsetter {
00239 private:
00240     /// container vector for plane waves in 1D
00241     std::vector<PlaneWave1D> planeWaves1D;
00242     /// container vector for plane waves in 2D
00243     std::vector<PlaneWave2D> planeWaves2D;
00244     /// container vector for plane waves in 3D
00245     std::vector<PlaneWave3D> planeWaves3D;
00246     /// container vector for Gaussian wave packets in 1D
00247     std::vector<Gauss1D> gauss1Ds;
00248     /// container vector for Gaussian wave packets in 2D
00249     std::vector<Gauss2D> gauss2Ds;
00250     /// container vector for Gaussian wave packets in 3D
00251     std::vector<Gauss3D> gauss3Ds;
00252
00253 public:
00254     /// function to set all coordinates to zero and then 'add' the field values
00255     void eval(unrealtype x, unrealtype y, unrealtype z,
00256               unrealtype *pTo6Space);
00257     /// function to fill the lattice space with initial field values
00258     // of all field vector containers
00259     void add(unrealtype x, unrealtype y, unrealtype z,
00260              unrealtype *pTo6Space);
00261     /// function to add plane waves in 1D to their container vector
00262     void addPlaneWave1D(std::array<sunrealtype, 3> k = {1, 0, 0},
00263                         std::array<sunrealtype, 3> p = {0, 0, 1},
00264                         std::array<sunrealtype, 3> phi = {0, 0, 0});
00265     /// function to add plane waves in 2D to their container vector
00266     void addPlaneWave2D(std::array<sunrealtype, 3> k = {1, 0, 0},
00267                         std::array<sunrealtype, 3> p = {0, 0, 1},
00268                         std::array<sunrealtype, 3> phi = {0, 0, 0});
00269     /// function to add plane waves in 3D to their container vector
00270     void addPlaneWave3D(std::array<sunrealtype, 3> k = {1, 0, 0},
00271                         std::array<sunrealtype, 3> p = {0, 0, 1},
00272                         std::array<sunrealtype, 3> phi = {0, 0, 0});
00273     /// function to add Gaussian wave packets in 1D to their container vector
00274     void addGauss1D(std::array<sunrealtype, 3> k = {1, 0, 0},
00275                      std::array<sunrealtype, 3> p = {0, 0, 1},
00276                      std::array<sunrealtype, 3> xo = {0, 0, 0},
00277                      unrealtype phig_ = 1.0,
00278                      std::array<sunrealtype, 3> phi = {0, 0, 0});
00279     /// function to add Gaussian wave packets in 2D to their container vector
00280     void addGauss2D(std::array<sunrealtype, 3> dis_ = {0, 0, 0},
00281                      std::array<sunrealtype, 3> axis_ = {1, 0, 0},
00282                      unrealtype Amp_ = 1.0, unrealtype phip_ = 0,
00283                      unrealtype w0_ = 1e-5, unrealtype zr_ = 4e-5,
00284                      unrealtype Ph0_ = 2e-5, unrealtype PhA_ = 0.45e-5);
00285     /// function to add Gaussian wave packets in 3D to their container vector
00286     void addGauss3D(std::array<sunrealtype, 3> dis_ = {0, 0, 0},
00287                      std::array<sunrealtype, 3> axis_ = {1, 0, 0},
00288                      unrealtype Amp_ = 1.0, unrealtype phip_ = 0,
00289                      unrealtype w0_ = 1e-5, unrealtype zr_ = 4e-5,
00290                      unrealtype Ph0_ = 2e-5, unrealtype PhA_ = 0.45e-5);
00291 };
00292

```

## 6.10 src/LatticePatch.cpp File Reference

Construction of the overall envelope lattice and the lattice patches.

```
#include "LatticePatch.h"
#include <math.h>
Include dependency graph for LatticePatch.cpp:
```



# Functions

- int **generatePatchwork** (const **Lattice** &envelopeLattice, **LatticePatch** &patchToMold, const int DLx, const int DLy, const int DLz)  
*Set up the patchwork.*
  - void **errorKill** (const std::string &errorMessage)  
*Print a specific error message to stderr.*
  - int **check\_error** (int error, const char \*funcname, int id)  
*helper function to check MPI errors*
  - int **check\_retval** (void \*returnvalue, const char \*funcname, int opt, int id)  
*helper function to check CVode errors*

### **6.10.1 Detailed Description**

Construction of the overall envelope lattice and the lattice patches.

Definition in file [LatticePatch.cpp](#).

## 6.10.2 Function Documentation

### 6.10.2.1 check\_error()

```
int check_error (
    int error,
    const char * funcname,
    int id )
```

helper function to check MPI errors

Check MPI errors. Error handler must be set.

Definition at line 928 of file [LatticePatch.cpp](#).

```
00929     int eclass, len;  
00930     char errorstring[MPI_MAX_ERROR_STRING];  
00931     if( error != MPI_SUCCESS ) {  
00932         MPI_Error_class(error,&eclass);  
00933         MPI_Error_string(error,errorstring,&len);  
00934         std::cerr << "MPI Error(process " << id << ") in " << funcname << " : "  
00935             << errorstring << ", from class " << eclass << std::endl;  
00936         return 1;  
00937     }  
00938     return 0;  
00939 }
```

### 6.10.2.2 check\_retval()

```
int check_retval (
    void * returnvalue,
    const char * funcname,
    int opt,
    int id )
```

helper function to check CVode errors

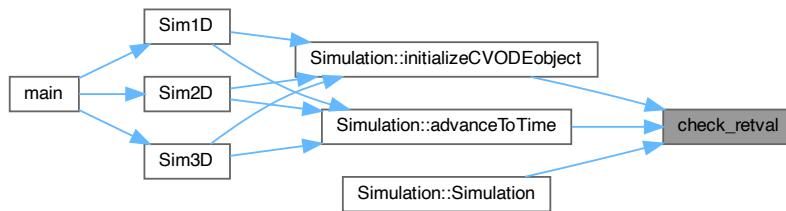
Check function return value. Adapted from CVode examples. opt == 0 means SUNDIALS function allocates memory so check if returned NULL pointer opt == 1 means SUNDIALS function returns an integer value so check if retval < 0 opt == 2 means function allocates memory so check if returned NULL pointer

Definition at line 948 of file [LatticePatch.cpp](#).

```
00948
00949     int *retval = nullptr;
00950
00951     /* Check if SUNDIALS function returned NULL pointer - no memory allocated */
00952     if (opt == 0 && returnvalue == nullptr) {
00953         fprintf(stderr,
00954             "\nSUNDIALS_ERROR(%d): %s() failed - returned NULL pointer\n\n", id,
00955             funcname);
00956         return (1);
00957     }
00958
00959     /* Check if retval < 0 */
00960     else if (opt == 1) {
00961         retval = (int *)returnvalue;
00962         char *flagname = CVodeGetReturnFlagName(*retval);
00963         if (*retval < 0) {
00964             fprintf(stderr, "\nSUNDIALS_ERROR(%d): %s() failed with retval = %d: "
00965                 "%s\n\n",
00966                 id, funcname, *retval, flagname);
00967             return (1);
00968         }
00969     }
00970
00971     /* Check if function returned NULL pointer - no memory allocated */
00972     else if (opt == 2 && returnvalue == nullptr) {
00973         fprintf(stderr,
00974             "\nMEMORY_ERROR(%d): %s() failed - returned NULL pointer\n\n", id,
00975             funcname);
00976         return (1);
00977     }
00978
00979     return (0);
00980 }
```

Referenced by [Simulation::advanceToTime\(\)](#), [Simulation::initializeCVODEobject\(\)](#), and [Simulation::Simulation\(\)](#).

Here is the caller graph for this function:



### 6.10.2.3 errorKill()

```
void errorKill (
    const std::string & errorMessage )
```

Print a specific error message to stderr.

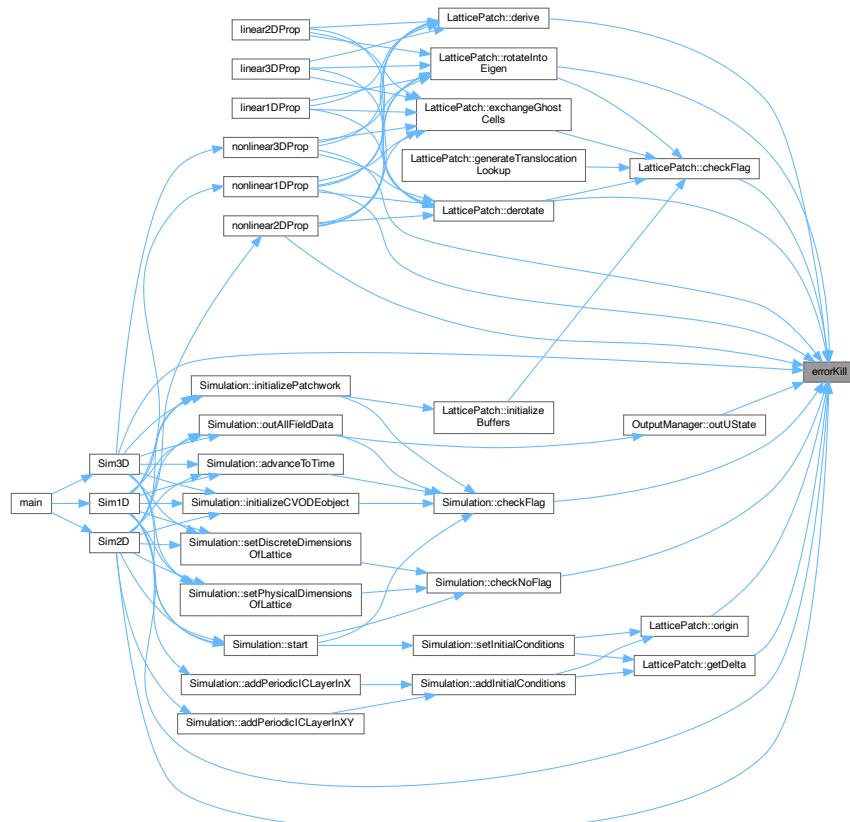
helper function for error messages

Definition at line 916 of file [LatticePatch.cpp](#).

```
00916     int my_prc=0;
00917     MPI_Comm_rank(MPI_COMM_WORLD,&my_prc);
00918     if (my_prc==0) {
00919         std::cerr << std::endl << "Error: " << errorMessage
00920         << "\nAborting..." << std::endl;
00921         MPI_Abort(MPI_COMM_WORLD, 1);
00922         return;
00923     }
00924 }
00925 }
```

Referenced by [LatticePatch::checkFlag\(\)](#), [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::getDelta\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), [nonlinear3DProp\(\)](#), [LatticePatch::origin\(\)](#), [OutputManager::outUState\(\)](#), [LatticePatch::rotateIntoEigen\(\)](#), [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



### 6.10.2.4 generatePatchwork()

```
int generatePatchwork (
    const Lattice & envelopeLattice,
    LatticePatch & patchToMold,
    const int DLx,
    const int DLy,
    const int DLz )
```

Set up the patchwork.

friend function for creating the patchwork slicing of the overall lattice

Definition at line 109 of file [LatticePatch.cpp](#).

```
00111 // Retrieve the ghost layer depth
00112 const int gLW = envelopeLattice.get_ghostLayerWidth();
00113 // Retrieve the data point dimension
00114 const int dPD = envelopeLattice.get_dataPointDimension();
00115 // MPI process/patch
00116 const int my_prc = envelopeLattice.my_prc;
00117 // Determine thicknes of the slice
00118 const sunindextype tot_NOXP = envelopeLattice.get_tot_nx();
00119 const sunindextype tot_NOYP = envelopeLattice.get_tot_ny();
00120 const sunindextype tot_NOZP = envelopeLattice.get_tot_nz();
00121 // position of the patch in the lattice of patches -> process associated to
00122 // position
00123 const sunindextype LIx = my_prc % DLx;
00124 const sunindextype LIy = (my_prc / DLx) % DLy;
00125 const sunindextype LIz = (my_prc / DLx) / DLy;
00126 // Determine the number of points in the patch and first absolute points in
00127 // each dimension
00128 const sunindextype local_NOXP = tot_NOXP / DLx;
00129 const sunindextype local_NOYP = tot_NOYP / DLy;
00130 const sunindextype local_NOZP = tot_NOZP / DLz;
00131 // absolute positions of the first point in each dimension
00132 const sunindextype firstXPoint = local_NOXP * LIx;
00133 const sunindextype firstYPoint = local_NOYP * LIy;
00134 const sunindextype firstZPoint = local_NOZP * LIz;
00135 // total number of points in the patch
00136 const sunindextype local_NODP = dPD * local_NOXP * local_NOYP * local_NOZP;
00137
00138 // Set patch up with above derived quantities
00139 patchToMold.dx = envelopeLattice.get_dx();
00140 patchToMold.dy = envelopeLattice.get_dy();
00141 patchToMold.dz = envelopeLattice.get_dz();
00142 patchToMold.x0 = firstXPoint * patchToMold.dx;
00143 patchToMold.y0 = firstYPoint * patchToMold.dy;
00144 patchToMold.z0 = firstZPoint * patchToMold.dz;
00145 patchToMold.LIx = LIx;
00146 patchToMold.LIy = LIy;
00147 patchToMold.LIz = LIz;
00148 patchToMold.nx = local_NOXP;
00149 patchToMold.ny = local_NOYP;
00150 patchToMold.nz = local_NOZP;
00151 patchToMold.lx = patchToMold.nx * patchToMold.dx;
00152 patchToMold.ly = patchToMold.ny * patchToMold.dy;
00153 patchToMold.lz = patchToMold.nz * patchToMold.dz;
00154
00155 #ifdef _OPENMP
00156 // OpenMP and MPI+X NVector interoperability
00157 // OpenMP NVectors with local patch size
00158 int num_threads = 1;
00159 num_threads = omp_get_max_threads();
00160 patchToMold.uLocal = N_VNew_OpenMP(local_NODP, num_threads,
00161 envelopeLattice.sunctx);
00162 patchToMold.duLocal = N_VNew_OpenMP(local_NODP, num_threads,
00163 envelopeLattice.sunctx);
00164 // MPI+X NVectors containing local OpenMP NVectors
00165 patchToMold.u = N_VMake_MPIPlusX(envelopeLattice.comm, patchToMold.uLocal,
00166 envelopeLattice.sunctx);
00167 patchToMold.du = N_VMake_MPIPlusX(envelopeLattice.comm, patchToMold.duLocal,
00168 envelopeLattice.sunctx);
00169 // Pointers to local vectors
00170 patchToMold.uData = N_VGetArrayPointer_MPIPlusX(patchToMold.u);
00171 patchToMold.duData = N_VGetArrayPointer_MPIPlusX(patchToMold.du);
00172
00173 #else
00174 // MPI NVectors with local patch and global lattice size
00175 patchToMold.u =
00176     N_VNew_Parallel(envelopeLattice.comm, local_NODP,
```

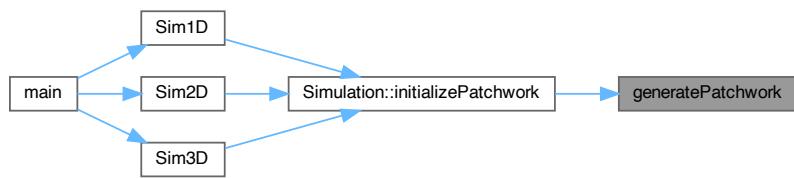
```

00177           envelopeLattice.get_tot_noDP(), envelopeLattice.sunctx);
00178   patchToMold.du =
00179     N_VNew_Parallel(envelopeLattice.comm, local_NODP,
00180                      envelopeLattice.get_tot_noDP(), envelopeLattice.sunctx);
00181   patchToMold.uData = NV_DATA_P(patchToMold.u);
00182   patchToMold.duData = NV_DATA_P(patchToMold.du);
00183 #endif
00184
00185 // Allocate space for auxiliary uAux so that the lattice and all possible
00186 // directions of ghost Layers fit
00187 const sunindextype s1 = patchToMold.nx, s2 = patchToMold.ny,
00188   s3 = patchToMold.nz;
00189 const sunindextype s_min = std::min(s1, std::min(s2, s3));
00190 patchToMold.uAux.resize(s1 * s2 * s3 / s_min * (s_min + 2 * gLW) * dPD);
00191 patchToMold.uAuxData = &patchToMold.uAux[0];
00192 patchToMold.envelopeLattice = &envelopeLattice;
00193 // Set patch "name" to process number -> only for debugging
00194 // patchToMold.ID=my_prc;
00195 // set flag
00196 patchToMold.statusFlags = FLatticePatchSetUp;
00197 patchToMold.generateTranslocationLookup();
00198 return 0;
00199 }

```

Referenced by [Simulation::initializePatchwork\(\)](#).

Here is the caller graph for this function:



## 6.11 LatticePatch.cpp

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file LatticePatch.cpp
00003 /// @brief Construction of the overall envelope lattice and the lattice patches
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #include "LatticePatch.h"
00007
00008 #include <math.h>
00009
00010 ///////////////////////////////////////////////////////////////////
00011 /// Implementation of Lattice component functions ///
00012 ///////////////////////////////////////////////////////////////////
00013
00014 /// Initialize the cartesian communicator
00015 void Lattice::initializeCommunicator(const int Nx, const int Ny,
00016   const int Nz, const bool per) {
00017   const int dims[3] = {Nz, Ny, Nx};
00018   const int periods[3] = {static_cast<int>(per), static_cast<int>(per),
00019     static_cast<int>(per)};
00020   // Create the cartesian communicator for MPI_COMM_WORLD
00021   MPI_Cart_create(MPI_COMM_WORLD, 3, dims, periods, 1, &comm);
00022   // Set MPI variables of the lattice
00023   MPI_Comm_size(comm, &(n_prc));
00024   MPI_Comm_rank(comm, &(my_prc));
00025   // Associate name to the communicator to identify it -> for debugging and
00026   // nicer error messages
00027   constexpr char lattice_comm_name[] = "Lattice";
00028   MPI_Comm_set_name(comm, lattice_comm_name);
00029 }
00030
00031 // Construct the lattice and set the stencil order

```

```

00032 Lattice::Lattice(const int Sto) : stencilOrder(Sto),
00033     ghostLayerWidth(Sto/2+1) {
00034     statusFlags = 0;
00035 }
00036
00037 /// Set the number of points in each dimension of the lattice
00038 void Lattice::setDiscreteDimensions(const sunindextype _nx,
00039     const sunindextype _ny, const sunindextype _nz) {
00040     // copy the given data for number of points
00041     tot_nx = _nx;
00042     tot_ny = _ny;
00043     tot_nz = _nz;
00044     // compute the resulting number of points and datapoints
00045     tot_noP = tot_nx * tot_ny * tot_nz;
00046     tot_noDP = dataPointDimension * tot_noP;
00047     // compute the new Delta, the physical resolution
00048     dx = tot_lx / tot_nx;
00049     dy = tot_ly / tot_ny;
00050     dz = tot_lz / tot_nz;
00051 }
00052
00053 /// Set the physical size of the lattice
00054 void Lattice::setPhysicalDimensions(const sunrealtype _lx,
00055     const sunrealtype _ly, const sunrealtype _lz) {
00056     tot_lx = _lx;
00057     tot_ly = _ly;
00058     tot_lz = _lz;
00059     // calculate physical distance between points
00060     dx = tot_lx / tot_nx;
00061     dy = tot_ly / tot_ny;
00062     dz = tot_lz / tot_nz;
00063     statusFlags |= FLatticeDimensionSet;
00064 }
00065
00066 ///////////////////////////////// Implementation of LatticePatch component functions //////////////////
00067 ///////////////////////////////// Implementation of LatticePatch component functions //////////////////
00068 ///////////////////////////////// Implementation of LatticePatch component functions //////////////////
00069
00070 /// Construct the lattice patch
00071 LatticePatch::LatticePatch() {
00072     // set default origin coordinates to (0,0,0)
00073     x0 = y0 = z0 = 0;
00074     // set default position in Lattice-Patchwork to (0,0,0)
00075     LIx = LIy = LIz = 0;
00076     // set default physical length for lattice patch to (0,0,0)
00077     lx = ly = lz = 0;
00078     // set default discrete length for lattice patch to (0,1,1)
00079     /* This is done in this manner as even in 1D simulations require a 1 point
00080      * width */
00081     nx = 0;
00082     ny = nz = 1;
00083
00084     // u is not initialized as it wouldn't make any sense before the dimensions
00085     // are set idem for the enveloping lattice
00086
00087     // set default statusFlags to non set
00088     statusFlags = 0;
00089 }
00090
00091 /// Destruct the patch and thereby destroy the NVectors
00092 LatticePatch::~LatticePatch() {
00093     // Deallocate memory for solution vector
00094     if (statusFlags & FLatticePatchSetUp) {
00095         // Destroy data vectors
00096 #if defined(_OPENMP)
00097         N_VDestroy(u);
00098         N_VDestroy(du);
00099         N_VDestroy_OpenMP(uLocal);
00100         N_VDestroy_OpenMP(duLocal);
00101 #else
00102         N_VDestroy_Parallel(u);
00103         N_VDestroy_Parallel(du);
00104 #endif
00105     }
00106 }
00107
00108 /// Set up the patchwork
00109 int generatePatchwork(const Lattice &envelopeLattice,
00110     LatticePatch &patchToMold,
00111     const int DLx, const int DLy, const int DLz) {
00112     // Retrieve the ghost layer depth
00113     const int gLW = envelopeLattice.get_ghostLayerWidth();
00114     // Retrieve the data point dimension
00115     const int dPD = envelopeLattice.get_dataPointDimension();
00116     // MPI process/patch
00117     const int my_prc = envelopeLattice.my_prc;
00118     // Determine thicknes of the slice

```

```

00119 const sunindextype tot_NOXP = envelopeLattice.get_tot_nx();
00120 const sunindextype tot_NOYP = envelopeLattice.get_tot_ny();
00121 const sunindextype tot_NOZP = envelopeLattice.get_tot_nz();
00122 // position of the patch in the lattice of patches -> process associated to
00123 // position
00124 const sunindextype LIx = my_prc % DLx;
00125 const sunindextype LIy = (my_prc / DLx) % DLy;
00126 const sunindextype LIz = (my_prc / DLx) / DLy;
00127 // Determine the number of points in the patch and first absolute points in
00128 // each dimension
00129 const sunindextype local_NOXP = tot_NOXP / DLx;
00130 const sunindextype local_NOYP = tot_NOYP / DLy;
00131 const sunindextype local_NOZP = tot_NOZP / DLz;
00132 // absolute positions of the first point in each dimension
00133 const sunindextype firstXPoint = local_NOXP * LIx;
00134 const sunindextype firstYPoint = local_NOYP * LIy;
00135 const sunindextype firstZPoint = local_NOZP * LIz;
00136 // total number of points in the patch
00137 const sunindextype local_NODP = dPD * local_NOXP * local_NOYP * local_NOZP;
00138
00139 // Set patch up with above derived quantities
00140 patchToMold.dx = envelopeLattice.get_dx();
00141 patchToMold.dy = envelopeLattice.get_dy();
00142 patchToMold.dz = envelopeLattice.get_dz();
00143 patchToMold.x0 = firstXPoint * patchToMold.dx;
00144 patchToMold.y0 = firstYPoint * patchToMold.dy;
00145 patchToMold.z0 = firstZPoint * patchToMold.dz;
00146 patchToMold.LIx = LIx;
00147 patchToMold.LIy = LIy;
00148 patchToMold.LIz = LIz;
00149 patchToMold.nx = local_NOXP;
00150 patchToMold.ny = local_NOYP;
00151 patchToMold.nz = local_NOZP;
00152 patchToMold.lx = patchToMold.nx * patchToMold.dx;
00153 patchToMold.ly = patchToMold.ny * patchToMold.dy;
00154 patchToMold.lz = patchToMold.nz * patchToMold.dz;
00155
00156 #ifdef _OPENMP
00157 // OpenMP and MPI+X NVectors interoperability
00158 // OpenMP NVectors with local patch size
00159 int num_threads = 1;
00160 num_threads = omp_get_max_threads();
00161 patchToMold.uLocal = N_VNew_OpenMP(local_NODP, num_threads,
00162     envelopeLattice.sunctx);
00163 patchToMold.duLocal = N_VNew_OpenMP(local_NODP, num_threads,
00164     envelopeLattice.sunctx);
00165 // MPI+X NVectors containing local OpenMP NVectors
00166 patchToMold.u = N_VMake_MPIPlusX(envelopeLattice.comm, patchToMold.uLocal,
00167     envelopeLattice.sunctx);
00168 patchToMold.du = N_VMake_MPIPlusX(envelopeLattice.comm, patchToMold.duLocal,
00169     envelopeLattice.sunctx);
00170 // Pointers to local vectors
00171 patchToMold.uData = N_VGetArrayPointer_MPIPlusX(patchToMold.u);
00172 patchToMold.duData = N_VGetArrayPointer_MPIPlusX(patchToMold.du);
00173 #else
00174 // MPI NVectors with local patch and global lattice size
00175 patchToMold.u =
00176     N_VNew_Parallel(envelopeLattice.comm, local_NODP,
00177         envelopeLattice.get_tot_noDP(), envelopeLattice.sunctx);
00178 patchToMold.du =
00179     N_VNew_Parallel(envelopeLattice.comm, local_NODP,
00180         envelopeLattice.get_tot_noDP(), envelopeLattice.sunctx);
00181 patchToMold.uData = NV_DATA_P(patchToMold.u);
00182 patchToMold.duData = NV_DATA_P(patchToMold.du);
00183 #endif
00184
00185 // Allocate space for auxiliary uAux so that the lattice and all possible
00186 // directions of ghost Layers fit
00187 const sunindextype s1 = patchToMold.nx, s2 = patchToMold.ny,
00188     s3 = patchToMold.nz;
00189 const sunindextype s_min = std::min(s1, std::min(s2, s3));
00190 patchToMold.uAux.resize(s1 * s2 * s3 / s_min * (s_min + 2 * gLW) * dPD);
00191 patchToMold.uAuxData = &patchToMold.uAux[0];
00192 patchToMold.envelopeLattice = &envelopeLattice;
00193 // Set patch "name" to process number -> only for debugging
00194 // patchToMold.ID=my_prc;
00195 // set flag
00196 patchToMold.statusFlags = FLatticePatchSetUp;
00197 patchToMold.generateTranslocationLookup();
00198 return 0;
00199 }
00200
00201 /// Return the discrete size of the patch: number of lattice patch points in
00202 /// specified dimension
00203 sunindextype LatticePatch::discreteSize(int dir) const {
00204     switch (dir) {
00205     case 0:

```

```

00206     return nx * ny * nz;
00207     case 1:
00208         return nx;
00209     case 2:
00210         return ny;
00211     case 3:
00212         return nz;
00213 // case 4: return uAux.size(); // for debugging
00214     default:
00215         return -1;
00216     }
00217 }
00218
00219 /// Return the physical origin of the patch in a dimension
00220 sunrealtype LatticePatch::origin(const int dir) const {
00221     switch (dir) {
00222     case 1:
00223         return x0;
00224     case 2:
00225         return y0;
00226     case 3:
00227         return z0;
00228     default:
00229         errorKill("LatticePatch::origin function called with wrong dir parameter");
00230         return -1;
00231     }
00232 }
00233
00234 /// Return the distance between points in the patch in a dimension
00235 sunrealtype LatticePatch::getDelta(const int dir) const {
00236     switch (dir) {
00237     case 1:
00238         return dx;
00239     case 2:
00240         return dy;
00241     case 3:
00242         return dz;
00243     default:
00244         errorKill(
00245             "LatticePatch::getDelta function called with wrong dir parameter");
00246         return -1;
00247     }
00248 }
00249
00250 /** In order to avoid cache misses:
00251 * create vectors to translate u vector into space coordinates and vice versa
00252 * and same for left and right ghost layers to space */
00253 void LatticePatch::generateTranslocationLookup() {
00254     // Check that the lattice has been set up
00255     checkFlag(FLatticeDimensionSet);
00256     // lengths for auxilliary layers, including ghost layers
00257     const int gLW = envelopeLattice->get_ghostLayerWidth();
00258     const sunindextype mx = nx + 2 * gLW;
00259     const sunindextype my = ny + 2 * gLW;
00260     const sunindextype mz = nz + 2 * gLW;
00261     // sizes for lookup vectors
00262     const sunindextype totalNP = nx * ny * nz;
00263     const sunindextype haloXSize = mx * ny * nz;
00264     const sunindextype haloYSize = nx * my * nz;
00265     const sunindextype haloZSize = nx * ny * mz;
00266     // generate u->uAux
00267     uTox.resize(totalNP);
00268     uToy.resize(totalNP);
00269     uToz.resize(totalNP);
00270     // generate uAux->u with length including halo
00271     xTou.resize(haloXSize);
00272     yTou.resize(haloYSize);
00273     zTou.resize(haloZSize);
00274     // same for ghost layer lookup tables
00275     const sunindextype ghostXSize = gLW * ny * nz;
00276     const sunindextype ghostYSize = gLW * nx * nz;
00277     const sunindextype ghostZSize = gLW * nx * ny;
00278     lgcTox.resize(ghostXSize);
00279     rgcTox.resize(ghostXSize);
00280     lgcToy.resize(ghostYSize);
00281     rgcToy.resize(ghostYSize);
00282     lgcToz.resize(ghostZSize);
00283     rgcToz.resize(ghostZSize);
00284     // variables for cartesian position in the 3D discrete lattice
00285     sunindextype px = 0, py = 0, pz = 0;
00286     // Fill the lookup tables
00287 #pragma omp parallel default(none) \
00288 private(px, py, pz) \
00289 shared(uTox, uToy, uToz, xTou, yTou, zTou, \
00290         nx, ny, mx, my, mz, gLW, totalNP, \
00291         lgcTox, rgcTox, lgcToy, rgcToy, lgcToz, rgcToz, \
00292         ghostXSize, ghostYSize, ghostZSize)

```

```

00293 {
00294 #pragma omp for simd schedule(static)
00295 for (sunindextype i = 0; i < totalNP; i++) { // loop over the patch
00296     // calculate cartesian coordinates
00297     px = i % nx;
00298     py = (i / nx) % ny;
00299     pz = (i / nx) / ny;
00300     // fill lookups extended by halos (useful for y and z direction)
00301     uTox[i] = (px + gLW) + py * mx +
00302                 pz * mx * ny; // unroll (de-flatten) cartesian dimension
00303     xTou[px + py * mx + pz * mx * ny] =
00304         i; // match cartesian point to u location
00305     uToy[i] = (py + gLW) + pz * my + px * my * nz;
00306     yTou[py + pz * my + px * my * nz] = i;
00307     uToz[i] = (pz + gLW) + px * mz + py * mz * nx;
00308     zTou[pz + px * mz + py * mz * nx] = i;
00309 }
00310 #pragma omp for simd schedule(static)
00311 for (sunindextype i = 0; i < ghostXSize; i++) {
00312     px = i % gLW;
00313     py = (i / gLW) % ny;
00314     pz = (i / gLW) / ny;
00315     lgcTox[i] = px + py * mx + pz * mx * ny;
00316     rgcTox[i] = px + ny + gLW + py * mx + pz * mx * ny;
00317 }
00318 #pragma omp for simd schedule(static)
00319 for (sunindextype i = 0; i < ghostYSize; i++) {
00320     px = i % nx;
00321     py = (i / nx) % gLW;
00322     pz = (i / nx) / gLW;
00323     lgcToy[i] = py + pz * my + px * my * nz;
00324     rgcToy[i] = py + ny + gLW + pz * my + px * my * nz;
00325 }
00326 #pragma omp for simd schedule(static)
00327 for (sunindextype i = 0; i < ghostZSize; i++) {
00328     px = i % nx;
00329     py = (i / nx) % ny;
00330     pz = (i / nx) / ny;
00331     lgcToz[i] = pz + px * mz + py * mz * nx;
00332     rgcToz[i] = pz + ny + gLW + px * mz + py * mz * nx;
00333 }
00334 }
00335 statusFlags |= TranslocationLookupSetUp;
00336 }
00337
00338 /** Rotate into eigenraum along R matrices of paper using the rotation
00339 * methods;
00340 * uAuxData gets the rotated left-halo-, inner-patch-, right-halo-data */
00341 void LatticePatch::rotateIntoEigen(const int dir) {
00342     // Check that the lattice, ghost layers as well as the translocation lookups
00343     // have been set up;
00344     checkFlag(FLatticePatchSetUp);
00345     checkFlag(TranslocationLookupSetUp);
00346     checkFlag(GhostLayersInitialized); // this check is only after call to
00347                                         // exchange ghost cells
00348     switch (dir) {
00349     case 1:
00350         rotateToX(uAuxData, gCLData, lgcTox);
00351         rotateToX(uAuxData, uData, uTox);
00352         rotateToX(uAuxData, gCRData, rgcTox);
00353         break;
00354     case 2:
00355         rotateToY(uAuxData, gCLData, lgcToy);
00356         rotateToY(uAuxData, uData, uToy);
00357         rotateToY(uAuxData, gCRData, rgcToy);
00358         break;
00359     case 3:
00360         rotateToZ(uAuxData, gCLData, lgcToz);
00361         rotateToZ(uAuxData, uData, uToz);
00362         rotateToZ(uAuxData, gCRData, rgcToz);
00363         break;
00364     default:
00365         errorKill("Tried to rotate into the wrong direction");
00366         break;
00367     }
00368 }
00369
00370 /// Rotate halo and inner-patch data vectors with rotation matrix Rx into
00371 /// eigenspace of Z matrix and write to auxiliary vector
00372 inline void LatticePatch::rotateToX(sunrealtype *outArray,
00373                                     const unrealtype *inArray,
00374                                     const std::vector<sunindextype> &lookup) {
00375     sunindextype ii = 0, target = 0;
00376     const sunindextype size = lookup.size();
00377     const int dPD = envelopeLattice->get_dataPointDimension();
00378     #pragma omp parallel for simd \
00379     private(target, ii) \

```

```

00380     shared(lookup, outArray, inArray, size, dPD) \
00381     schedule(static)
00382     for (sunindextype i = 0; i < size; i++) {
00383         // get correct u-vector and spatial indices along previously defined lookup
00384         // tables
00385         target = dPD * lookup[i];
00386         ii = dPD * i;
00387         outArray[target + 0] = -inArray[1 + ii] + inArray[5 + ii];
00388         outArray[target + 1] = inArray[2 + ii] + inArray[4 + ii];
00389         outArray[target + 2] = inArray[1 + ii] + inArray[5 + ii];
00390         outArray[target + 3] = -inArray[2 + ii] + inArray[4 + ii];
00391         outArray[target + 4] = inArray[3 + ii];
00392         outArray[target + 5] = inArray[ii];
00393     }
00394 }
00395
00396 /// Rotate halo and inner-patch data vectors with rotation matrix Ry into
00397 /// eigenspace of Z matrix and write to auxiliary vector
00398 inline void LatticePatch::rotateToY(sunrealtype *outArray,
00399                                         const unrealtype *inArray,
00400                                         const std::vector<sunindextype> &lookup) {
00401     sunindextype ii = 0, target = 0;
00402     const int dPD = envelopeLattice->get_dataPointDimension();
00403     const sunindextype size = lookup.size();
00404     #pragma omp parallel for simd \
00405     private(target, ii) \
00406     shared(lookup, outArray, inArray, size, dPD) \
00407     schedule(static)
00408     for (sunindextype i = 0; i < size; i++) {
00409         target = dPD * lookup[i];
00410         ii = dPD * i;
00411         outArray[target + 0] = inArray[ii] + inArray[5 + ii];
00412         outArray[target + 1] = -inArray[2 + ii] + inArray[3 + ii];
00413         outArray[target + 2] = -inArray[ii] + inArray[5 + ii];
00414         outArray[target + 3] = inArray[2 + ii] + inArray[3 + ii];
00415         outArray[target + 4] = inArray[4 + ii];
00416         outArray[target + 5] = inArray[1 + ii];
00417     }
00418 }
00419
00420 /// Rotate halo and inner-patch data vectors with rotation matrix Rz into
00421 /// eigenspace of Z matrix and write to auxiliary vector
00422 inline void LatticePatch::rotateToZ(sunrealtype *outArray,
00423                                         const unrealtype *inArray,
00424                                         const std::vector<sunindextype> &lookup) {
00425     sunindextype ii = 0, target = 0;
00426     const sunindextype size = lookup.size();
00427     const int dPD = envelopeLattice->get_dataPointDimension();
00428     #pragma omp parallel for simd \
00429     private(target, ii) \
00430     shared(lookup, outArray, inArray, size, dPD) \
00431     schedule(static)
00432     for (sunindextype i = 0; i < size; i++) {
00433         target = dPD * lookup[i];
00434         ii = dPD * i;
00435         outArray[target + 0] = -inArray[ii] + inArray[4 + ii];
00436         outArray[target + 1] = inArray[1 + ii] + inArray[3 + ii];
00437         outArray[target + 2] = inArray[ii] + inArray[4 + ii];
00438         outArray[target + 3] = -inArray[1 + ii] + inArray[3 + ii];
00439         outArray[target + 4] = inArray[5 + ii];
00440         outArray[target + 5] = inArray[2 + ii];
00441     }
00442 }
00443
00444 /// Derotate uAux with transposed rotation matrices and write to derivative
00445 /// buffer -- normalization is done here by the factor 1/2
00446 void LatticePatch::derotate(int dir, unrealtype *buffOut) {
00447     // Check that the lattice as well as the translocation lookups have been set
00448     // up;
00449     checkFlag(FLatticePatchSetUp);
00450     checkFlag(TranslocationLookupSetUp);
00451     const int dPD = envelopeLattice->get_dataPointDimension();
00452     const int gLW = envelopeLattice->get_ghostLayerWidth();
00453     const sunindextype totalNP = discreteSize();
00454     sunindextype ii = 0, target = 0;
00455     switch (dir) {
00456     case 1:
00457         #pragma omp parallel for simd \
00458         private(ii, target) \
00459         shared(dPD, gLW, totalNP, uTox, uAux, buffOut) \
00460         schedule(static)
00461         for (sunindextype i = 0; i < totalNP; i++) {
00462             // get correct indices in u and rotation space
00463             target = dPD * i;
00464             ii = dPD * (uTox[i] - gLW);
00465             buffOut[target + 0] = uAux[5 + ii];
00466             buffOut[target + 1] = (-uAux[ii] + uAux[2 + ii]) / 2.;

```

```

00467     buffOut[target + 2] = (uAux[1 + ii] - uAux[3 + ii]) / 2.;
00468     buffOut[target + 3] = uAux[4 + ii];
00469     buffOut[target + 4] = (uAux[1 + ii] + uAux[3 + ii]) / 2.;
00470     buffOut[target + 5] = (uAux[ii] + uAux[2 + ii]) / 2.;
00471 }
00472 break;
00473 case 2:
00474 #pragma omp parallel for simd \
00475 private(ii, target) \
00476 shared(dPD, gLW, totalNP, uTox, uAux, buffOut) \
00477 schedule(static)
00478 for (sunindextype i = 0; i < totalNP; i++) {
00479     target = dPD * i;
00480     ii = dPD * (uToy[i] - gLW);
00481     buffOut[target + 0] = (uAux[ii] - uAux[2 + ii]) / 2.;
00482     buffOut[target + 1] = uAux[5 + ii];
00483     buffOut[target + 2] = (-uAux[1 + ii] + uAux[3 + ii]) / 2.;
00484     buffOut[target + 3] = (uAux[1 + ii] + uAux[3 + ii]) / 2.;
00485     buffOut[target + 4] = uAux[4 + ii];
00486     buffOut[target + 5] = (uAux[ii] + uAux[2 + ii]) / 2.;
00487 }
00488 break;
00489 case 3:
00490 #pragma omp parallel for simd \
00491 private(ii, target) \
00492 shared(dPD, gLW, totalNP, uTox, uAux, buffOut) \
00493 schedule(static)
00494 for (sunindextype i = 0; i < totalNP; i++) {
00495     target = dPD * i;
00496     ii = dPD * (uToz[i] - gLW);
00497     buffOut[target + 0] = (-uAux[ii] + uAux[2 + ii]) / 2.;
00498     buffOut[target + 1] = (uAux[1 + ii] - uAux[3 + ii]) / 2.;
00499     buffOut[target + 2] = uAux[5 + ii];
00500     buffOut[target + 3] = (uAux[1 + ii] + uAux[3 + ii]) / 2.;
00501     buffOut[target + 4] = (uAux[ii] + uAux[2 + ii]) / 2.;
00502     buffOut[target + 5] = uAux[4 + ii];
00503 }
00504 break;
00505 default:
00506     errorKill("Tried to derotate from the wrong direction");
00507     break;
00508 }
00509 }
00510
00511 /// Create buffers to save derivative values, optimizing computational load
00512 void LatticePatch::initializeBuffers() {
00513     // Check that the lattice has been set up
00514     checkFlag(FLatticeDimensionSet);
00515     const int dPD = envelopeLattice->get_dataPointDimension();
00516     buffX.resize(nx * ny * nz * dPD);
00517     buffY.resize(nx * ny * nz * dPD);
00518     buffZ.resize(nx * ny * nz * dPD);
00519     // Set pointers used for propagation functions
00520     buffData[0] = &buffX[0];
00521     buffData[1] = &buffY[0];
00522     buffData[2] = &buffZ[0];
00523     statusFlags |= BuffersInitialized;
00524 }
00525
00526 /// Perform the ghost cell exchange in a specified direction
00527 void LatticePatch::exchangeGhostCells(const int dir) {
00528     // Check that the lattice has been set up
00529     checkFlag(FLatticeDimensionSet);
00530     checkFlag(FLatticePatchSetUp);
00531     // Variables to per dimension calculate the halo indices, and distance to
00532     // other side halo boundary
00533     int mx = 1, my = 1, mz = 1, distToLeft = 1;
00534     const int gLW = envelopeLattice->get_ghostLayerWidth();
00535     // In the chosen direction m is set to ghost layer width while the others
00536     // remain to form the plane
00537     switch (dir) {
00538     case 1:
00539         mx = gLW;
00540         my = ny;
00541         mz = nz;
00542         distToLeft = (nx - gLW);
00543         break;
00544     case 2:
00545         mx = nx;
00546         my = gLW;
00547         mz = nz;
00548         distToLeft = nx * (ny - gLW);
00549         break;
00550     case 3:
00551         mx = nx;
00552         my = ny;
00553         mz = gLW;

```

```

00554     distToRight = nx * ny * (nz - gLW);
00555     break;
00556 }
00557 // total number of exchanged points
00558 const int dPD = envelopeLattice->get_dataPointDimension();
00559 const sunindextype exchangeSize = mx * my * mz * dPD;
00560 // provide size of the halos for ghost cells
00561 ghostCellLeft.resize(exchangeSize);
00562 ghostCellRight.resize(ghostCellLeft.size());
00563 ghostCellLeftToSend.resize(ghostCellLeft.size());
00564 ghostCellRightToSend.resize(ghostCellLeft.size());
00565 gCLData = &ghostCellLeft[0];
00566 gCRData = &ghostCellRight[0];
00567 statusFlags |= GhostLayersInitialized;
00568
00569 // Initialize running index li for the halo buffers, and index ui of uData for
00570 // data transfer
00571 sunindextype li = 0, ui = 0;
00572 // Fill the halo buffers
00573 #pragma omp parallel for default(none) \
00574 private(ui, li) \
00575 shared(nx, ny, mx, my, mz, dPD, distToRight, uData, \
00576         ghostCellLeftToSend, ghostCellRightToSend)
00577 for (sunindextype iz = 0; iz < mz; iz++) {
00578     for (sunindextype iy = 0; iy < my; iy++) {
00579         // uData vector start index of halo data to be transferred
00580         // with each z-step add the whole xy-plane and with y-step the x-range ->
00581         // iterate all x-ranges
00582         ui = (iz * nx * ny + iy * nx) * dPD;
00583         // increase halo index by transferred items of previous iteration steps
00584         li = (iz * my * mx + iy * mx) * dPD;
00585         // copy left halo data from uData to buffer, transfer size is given by
00586         // x-length (not x-range)
00587         std::copy(&uData[ui], &uData[ui + mx * dPD], &ghostCellLeftToSend[li]);
00588         ui += distToRight * dPD;
00589         std::copy(&uData[ui], &uData[ui + mx * dPD], &ghostCellRightToSend[li]);
00590     }
00591 }
00592
00593 /* Send and receive the data to and from neighboring latticePatches */
00594 // Adjust direction to cartesian communicator
00595 int dim = 2; // default for dir==1
00596 if (dir == 2) {
00597     dim = 1;
00598 } else if (dir == 3) {
00599     dim = 0;
00600 }
00601 int rank_source = 0, rank_dest = 0;
00602 MPI_Cart_shift(envelopeLattice->comm, dim, -1, &rank_source,
00603                 &rank_dest); // s.t. rank_dest is left & v.v.
00604
00605 // nonblocking Irecv/Isend
00606
00607 MPI_Request requests[4];
00608 MPI_Irecv(&ghostCellRight[0], exchangeSize, MPI_SUNREALTYPE, rank_source, 1,
00609 envelopeLattice->comm, &requests[0]);
00610 MPI_Isend(&ghostCellLeftToSend[0], exchangeSize, MPI_SUNREALTYPE, rank_dest,
00611 1, envelopeLattice->comm, &requests[1]);
00612 MPI_Irecv(&ghostCellLeft[0], exchangeSize, MPI_SUNREALTYPE, rank_dest, 2,
00613 envelopeLattice->comm, &requests[2]);
00614 MPI_Isend(&ghostCellRightToSend[0], exchangeSize, MPI_SUNREALTYPE,
00615 rank_source, 2, envelopeLattice->comm, &requests[3]);
00616 MPI_Waitall(4, requests, MPI_STATUS_IGNORE);
00617
00618
00619 // blocking Sendrecv:
00620 /*
00621 MPI_Sendrecv(&ghostCellLeftToSend[0], exchangeSize, MPI_SUNREALTYPE,
00622             rank_dest, 1, &ghostCellRight[0], exchangeSize, MPI_SUNREALTYPE,
00623             rank_source, 1, envelopeLattice->comm, MPI_STATUS_IGNORE);
00624 MPI_Sendrecv(&ghostCellRightToSend[0], exchangeSize, MPI_SUNREALTYPE,
00625             rank_source, 2, &ghostCellLeft[0], exchangeSize, MPI_SUNREALTYPE,
00626             rank_dest, 2, envelopeLattice->comm, MPI_STATUS_IGNORE);
00627 */
00628 }
00629
00630 /// Check if all flags are set
00631 void LatticePatch::checkFlag(unsigned int flag) const {
00632     if (!(statusFlags & flag)) {
00633         std::string errorMessage;
00634         switch (flag) {
00635             case FLatticePatchSetUp:
00636                 errorMessage = "The Lattice patch was not set up please make sure to "
00637                             "initialize a Lattice topology";
00638                 break;
00639             case TranslocationLookupSetUp:
00640                 errorMessage = "The translocation lookup tables have not been generated, "

```

```

00641         "please be sure to run generateTranslocationLookup();";
00642         break;
00643     case GhostLayersInitialized:
00644         errorMessage = "The space for the ghost layers has not been allocated, "
00645             "please be sure that the ghost cells are initialized ";
00646         break;
00647     case BuffersInitialized:
00648         errorMessage = "The space for the buffers has not been allocated, please "
00649             "be sure to run initializeBuffers();";
00650         break;
00651     default:
00652         errorMessage = "Uppss, you've made a non-standard error, sadly I can't "
00653             "help you there";
00654         break;
00655     }
00656     errorKill(errorMessage);
00657 }
00658 return;
00659 }

00660 /// Calculate derivatives in the patch (uAux) in the specified direction
00661 void LatticePatch::derive(const int dir) {
00662     // ghost layer width adjusted to the chosen stencil order
00663     const int gLW = envelopeLattice->get_ghostLayerWidth();
00664     // dimensionality of data points -> 6
00665     const int dPD = envelopeLattice->get_dataPointDimension();
00666     // total width of patch in given direction including ghost layers at ends
00667     const sunindextype dirWidth = discreteSize(dir) + 2 * gLW;
00668     // width of patch only in given direction
00669     const sunindextype dirWidthO = discreteSize(dir);
00670     // size of plane perpendicular to given dimension
00671     const sunindextype perpPlainSize = discreteSize() / discreteSize(dir);
00672     // physical distance between points in that direction
00673     sunrealtype dxi = nan("0x12345");
00674     switch (dir) {
00675     case 1:
00676         dxi = dx;
00677         break;
00678     case 2:
00679         dxi = dy;
00680         break;
00681     case 3:
00682         dxi = dz;
00683         break;
00684     default:
00685         dxi = 1;
00686         errorKill("Tried to derive in the wrong direction");
00687         break;
00688     }
00689     // Derive according to chosen stencil accuracy order
00690     const int order = envelopeLattice->get_stencilOrder();
00691     switch (order) {
00692     case 1: // gLW=1
00693         #pragma omp parallel for default(none) \
00694             shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00695         for (sunindextype i = 0; i < perpPlainSize; i++) {
00696             #pragma omp simd
00697             for (sunindextype j = (i * dirWidth + gLW) * dPD;
00698                 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00699                 uAux[j + 0 - gLW * dPD] = slb(&uAux[j + 0]) / dxi;
00700                 uAux[j + 1 - gLW * dPD] = slb(&uAux[j + 1]) / dxi;
00701                 uAux[j + 2 - gLW * dPD] = slf(&uAux[j + 2]) / dxi;
00702                 uAux[j + 3 - gLW * dPD] = slf(&uAux[j + 3]) / dxi;
00703                 uAux[j + 4 - gLW * dPD] = slf(&uAux[j + 4]) / dxi;
00704                 uAux[j + 5 - gLW * dPD] = slf(&uAux[j + 5]) / dxi;
00705             }
00706         }
00707     }
00708     break;
00709     case 2: // gLW=2
00710         #pragma omp parallel for default(none) \
00711             shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)
00712         for (sunindextype i = 0; i < perpPlainSize; i++) {
00713             #pragma omp simd
00714             for (sunindextype j = (i * dirWidth + gLW) * dPD;
00715                 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00716                 uAux[j + 0 - gLW * dPD] = s2b(&uAux[j + 0]) / dxi;
00717                 uAux[j + 1 - gLW * dPD] = s2b(&uAux[j + 1]) / dxi;
00718                 uAux[j + 2 - gLW * dPD] = s2f(&uAux[j + 2]) / dxi;
00719                 uAux[j + 3 - gLW * dPD] = s2f(&uAux[j + 3]) / dxi;
00720                 uAux[j + 4 - gLW * dPD] = s2c(&uAux[j + 4]) / dxi;
00721                 uAux[j + 5 - gLW * dPD] = s2c(&uAux[j + 5]) / dxi;
00722             }
00723         }
00724     break;
00725     case 3: // gLW=2
00726         #pragma omp parallel for default(none) \
00727             shared(perpPlainSize, dxi, dirWidth, dirWidthO, gLW, dPD, uAux)

```

```

00728     for (sunindextype i = 0; i < perpPlainSize; i++) {
00729         #pragma omp simd
00730         for (sunindextype j = (i * dirWidth + gLW) * dPD;
00731             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00732             uAux[j + 0 - gLW * dPD] = s3b(&uAux[j + 0]) / dxii;
00733             uAux[j + 1 - gLW * dPD] = s3b(&uAux[j + 1]) / dxii;
00734             uAux[j + 2 - gLW * dPD] = s3f(&uAux[j + 2]) / dxii;
00735             uAux[j + 3 - gLW * dPD] = s3f(&uAux[j + 3]) / dxii;
00736             uAux[j + 4 - gLW * dPD] = s3f(&uAux[j + 4]) / dxii;
00737             uAux[j + 5 - gLW * dPD] = s3f(&uAux[j + 5]) / dxii;
00738         }
00739     }
00740     break;
00741 case 4: // gLW=3
00742     #pragma omp parallel for default(none) \
00743     shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00744     for (sunindextype i = 0; i < perpPlainSize; i++) {
00745         #pragma omp simd
00746         for (sunindextype j = (i * dirWidth + gLW) * dPD;
00747             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00748             uAux[j + 0 - gLW * dPD] = s4b(&uAux[j + 0]) / dxii;
00749             uAux[j + 1 - gLW * dPD] = s4b(&uAux[j + 1]) / dxii;
00750             uAux[j + 2 - gLW * dPD] = s4f(&uAux[j + 2]) / dxii;
00751             uAux[j + 3 - gLW * dPD] = s4f(&uAux[j + 3]) / dxii;
00752             uAux[j + 4 - gLW * dPD] = s4c(&uAux[j + 4]) / dxii;
00753             uAux[j + 5 - gLW * dPD] = s4c(&uAux[j + 5]) / dxii;
00754         }
00755     }
00756     break;
00757 case 5: // gLW=3
00758     #pragma omp parallel for default(none) \
00759     shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00760     for (sunindextype i = 0; i < perpPlainSize; i++) {
00761         #pragma omp simd
00762         for (sunindextype j = (i * dirWidth + gLW) * dPD;
00763             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00764             uAux[j + 0 - gLW * dPD] = s5b(&uAux[j + 0]) / dxii;
00765             uAux[j + 1 - gLW * dPD] = s5b(&uAux[j + 1]) / dxii;
00766             uAux[j + 2 - gLW * dPD] = s5f(&uAux[j + 2]) / dxii;
00767             uAux[j + 3 - gLW * dPD] = s5f(&uAux[j + 3]) / dxii;
00768             uAux[j + 4 - gLW * dPD] = s5f(&uAux[j + 4]) / dxii;
00769             uAux[j + 5 - gLW * dPD] = s5f(&uAux[j + 5]) / dxii;
00770         }
00771     }
00772     break;
00773 case 6: // gLW=4
00774     #pragma omp parallel for default(none) \
00775     shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00776     for (sunindextype i = 0; i < perpPlainSize; i++) {
00777         #pragma omp simd
00778         for (sunindextype j = (i * dirWidth + gLW) * dPD;
00779             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00780             uAux[j + 0 - gLW * dPD] = s6b(&uAux[j + 0]) / dxii;
00781             uAux[j + 1 - gLW * dPD] = s6b(&uAux[j + 1]) / dxii;
00782             uAux[j + 2 - gLW * dPD] = s6f(&uAux[j + 2]) / dxii;
00783             uAux[j + 3 - gLW * dPD] = s6f(&uAux[j + 3]) / dxii;
00784             uAux[j + 4 - gLW * dPD] = s6c(&uAux[j + 4]) / dxii;
00785             uAux[j + 5 - gLW * dPD] = s6c(&uAux[j + 5]) / dxii;
00786         }
00787     }
00788     break;
00789 case 7: // gLW=4
00790     #pragma omp parallel for default(none) \
00791     shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00792     for (sunindextype i = 0; i < perpPlainSize; i++) {
00793         #pragma omp simd
00794         for (sunindextype j = (i * dirWidth + gLW) * dPD;
00795             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00796             uAux[j + 0 - gLW * dPD] = s7b(&uAux[j + 0]) / dxii;
00797             uAux[j + 1 - gLW * dPD] = s7b(&uAux[j + 1]) / dxii;
00798             uAux[j + 2 - gLW * dPD] = s7f(&uAux[j + 2]) / dxii;
00799             uAux[j + 3 - gLW * dPD] = s7f(&uAux[j + 3]) / dxii;
00800             uAux[j + 4 - gLW * dPD] = s7f(&uAux[j + 4]) / dxii;
00801             uAux[j + 5 - gLW * dPD] = s7f(&uAux[j + 5]) / dxii;
00802         }
00803     }
00804     break;
00805 case 8: // gLW=5
00806     #pragma omp parallel for default(none) \
00807     shared(perpPlainSize, dxii, dirWidth, dirWidthO, gLW, dPD, uAux)
00808     for (sunindextype i = 0; i < perpPlainSize; i++) {
00809         #pragma omp simd
00810         for (sunindextype j = (i * dirWidth + gLW) * dPD;
00811             j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00812             uAux[j + 0 - gLW * dPD] = s8b(&uAux[j + 0]) / dxii;
00813             uAux[j + 1 - gLW * dPD] = s8b(&uAux[j + 1]) / dxii;
00814             uAux[j + 2 - gLW * dPD] = s8f(&uAux[j + 2]) / dxii;

```

```

00815     uAux[j + 3 - gLW * dPD] = s8f(&uAux[j + 3]) / dx;
00816     uAux[j + 4 - gLW * dPD] = s8c(&uAux[j + 4]) / dx;
00817     uAux[j + 5 - gLW * dPD] = s8c(&uAux[j + 5]) / dx;
00818 }
00819 }
00820 break;
00821 case 9: // gLW=5
00822 #pragma omp parallel for default(none) \
00823 shared(perpPlainSize, dx, dirWidth, dirWidthO, gLW, dPD, uAux)
00824 for (sunindextype i = 0; i < perpPlainSize; i++) {
00825 #pragma omp simd
00826 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00827 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00828 uAux[j + 0 - gLW * dPD] = s9b(&uAux[j + 0]) / dx;
00829 uAux[j + 1 - gLW * dPD] = s9b(&uAux[j + 1]) / dx;
00830 uAux[j + 2 - gLW * dPD] = s9f(&uAux[j + 2]) / dx;
00831 uAux[j + 3 - gLW * dPD] = s9f(&uAux[j + 3]) / dx;
00832 uAux[j + 4 - gLW * dPD] = s9f(&uAux[j + 4]) / dx;
00833 uAux[j + 5 - gLW * dPD] = s9f(&uAux[j + 5]) / dx;
00834 }
00835 }
00836 break;
00837 case 10: // gLW=6
00838 #pragma omp parallel for default(none) \
00839 shared(perpPlainSize, dx, dirWidth, dirWidthO, gLW, dPD, uAux)
00840 for (sunindextype i = 0; i < perpPlainSize; i++) {
00841 #pragma omp simd
00842 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00843 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00844 uAux[j + 0 - gLW * dPD] = s10b(&uAux[j + 0]) / dx;
00845 uAux[j + 1 - gLW * dPD] = s10b(&uAux[j + 1]) / dx;
00846 uAux[j + 2 - gLW * dPD] = s10f(&uAux[j + 2]) / dx;
00847 uAux[j + 3 - gLW * dPD] = s10f(&uAux[j + 3]) / dx;
00848 uAux[j + 4 - gLW * dPD] = s10c(&uAux[j + 4]) / dx;
00849 uAux[j + 5 - gLW * dPD] = s10c(&uAux[j + 5]) / dx;
00850 }
00851 }
00852 break;
00853 case 11: // gLW=6
00854 #pragma omp parallel for default(none) \
00855 shared(perpPlainSize, dx, dirWidth, dirWidthO, gLW, dPD, uAux)
00856 for (sunindextype i = 0; i < perpPlainSize; i++) {
00857 #pragma omp simd
00858 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00859 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00860 uAux[j + 0 - gLW * dPD] = s11b(&uAux[j + 0]) / dx;
00861 uAux[j + 1 - gLW * dPD] = s11b(&uAux[j + 1]) / dx;
00862 uAux[j + 2 - gLW * dPD] = s11f(&uAux[j + 2]) / dx;
00863 uAux[j + 3 - gLW * dPD] = s11f(&uAux[j + 3]) / dx;
00864 uAux[j + 4 - gLW * dPD] = s11f(&uAux[j + 4]) / dx;
00865 uAux[j + 5 - gLW * dPD] = s11f(&uAux[j + 5]) / dx;
00866 }
00867 }
00868 break;
00869 case 12: // gLW=7
00870 #pragma omp parallel for default(none) \
00871 shared(perpPlainSize, dx, dirWidth, dirWidthO, gLW, dPD, uAux)
00872 for (sunindextype i = 0; i < perpPlainSize; i++) {
00873 #pragma omp simd
00874 for (sunindextype j = (i * dirWidth + gLW) * dPD;
00875 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00876 uAux[j + 0 - gLW * dPD] = s12b(&uAux[j + 0]) / dx;
00877 uAux[j + 1 - gLW * dPD] = s12b(&uAux[j + 1]) / dx;
00878 uAux[j + 2 - gLW * dPD] = s12f(&uAux[j + 2]) / dx;
00879 uAux[j + 3 - gLW * dPD] = s12f(&uAux[j + 3]) / dx;
00880 uAux[j + 4 - gLW * dPD] = s12c(&uAux[j + 4]) / dx;
00881 uAux[j + 5 - gLW * dPD] = s12c(&uAux[j + 5]) / dx;
00882 }
00883 }
00884 break;
00885 case 13: // gLW=7
00886 // For all points in the plane perpendicular to the given direction
00887 #pragma omp parallel for default(none) \
00888 shared(perpPlainSize, dx, dirWidth, dirWidthO, gLW, dPD, uAux)
00889 for (sunindextype i = 0; i < perpPlainSize; i++) {
00890 // iterate through the derivation direction
00891 #pragma omp simd
00892 for (sunindextype j = (i * dirWidth
00893 + gLW /*to shift left by gLW below*/) * dPD;
00894 j < (i * dirWidth + gLW + dirWidthO) * dPD; j += dPD) {
00895 // Compute the stencil derivative for any of the six field components
00896 // and update position by ghost width shift
00897 uAux[j + 0 - gLW * dPD] = s13b(&uAux[j + 0]) / dx;
00898 uAux[j + 1 - gLW * dPD] = s13b(&uAux[j + 1]) / dx;
00899 uAux[j + 2 - gLW * dPD] = s13f(&uAux[j + 2]) / dx;
00900 uAux[j + 3 - gLW * dPD] = s13f(&uAux[j + 3]) / dx;
00901 uAux[j + 4 - gLW * dPD] = s13f(&uAux[j + 4]) / dx;

```

```

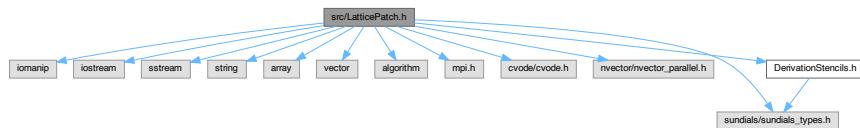
00902         uAux[j + 5 - gLW * dPD] = s13f(&uAux[j + 5]) / dx;
00903     }
00904 }
00905 break;
00906
00907 default:
00908     errorKill("Please set an existing stencil order");
00909     break;
00910 }
00911 }
00912
00913 ////////////// Helper functions ///////////
00914
00915 // Print a specific error message to stderr
00916 void errorKill(const std::string & errorMessage) {
00917     int my_prc=0;
00918     MPI_Comm_rank(MPI_COMM_WORLD,&my_prc);
00919     if (my_prc==0) {
00920         std::cerr << std::endl << "Error: " << errorMessage
00921         << "\nAborting..." << std::endl;
00922     MPI_Abort(MPI_COMM_WORLD, 1);
00923     return;
00924 }
00925 }
00926
00927 /** Check MPI errors. Error handler must be set. */
00928 int check_error(int error, const char *funcname, int id) {
00929     int eclass, len;
00930     char errorstring[MPI_MAX_ERROR_STRING];
00931     if(error != MPI_SUCCESS) {
00932         MPI_Error_class(error,&eclass);
00933         MPI_Error_string(error,errorstring,&len);
00934         std::cerr << "MPI Error(process " << id << ") in " << funcname << " : "
00935         << errorstring << ", from class " << eclass << std::endl;
00936     return 1;
00937 }
00938 return 0;
00939 }
00940
00941 /** Check function return value. Adapted from CVode examples.
00942     opt == 0 means SUNDIALS function allocates memory so check if
00943         returned NULL pointer
00944     opt == 1 means SUNDIALS function returns an integer value so check if
00945         retval < 0
00946     opt == 2 means function allocates memory so check if returned
00947         NULL pointer */
00948 int check_retval(void *returnvalue, const char *funcname, int opt, int id) {
00949     int *retval = nullptr;
00950
00951     /* Check if SUNDIALS function returned NULL pointer - no memory allocated */
00952     if (opt == 0 && returnvalue == nullptr) {
00953         fprintf(stderr,
00954             "\nSUNDIALS_ERROR(%d): %s() failed - returned NULL pointer\n\n", id,
00955             funcname);
00956         return (1);
00957     }
00958
00959     /* Check if retval < 0 */
00960     else if (opt == 1) {
00961         retval = (int *)returnvalue;
00962         char *flagname = CVodeGetReturnFlagName(*retval);
00963         if (*retval < 0) {
00964             fprintf(stderr, "\nSUNDIALS_ERROR(%d): %s() failed with retval = %d: "
00965                 "%s\n\n",
00966                 id, funcname, *retval, flagname);
00967             return (1);
00968         }
00969     }
00970
00971     /* Check if function returned NULL pointer - no memory allocated */
00972     else if (opt == 2 && returnvalue == nullptr) {
00973         fprintf(stderr,
00974             "\nMEMORY_ERROR(%d): %s() failed - returned NULL pointer\n\n", id,
00975             funcname);
00976         return (1);
00977     }
00978
00979     return (0);
00980 }

```

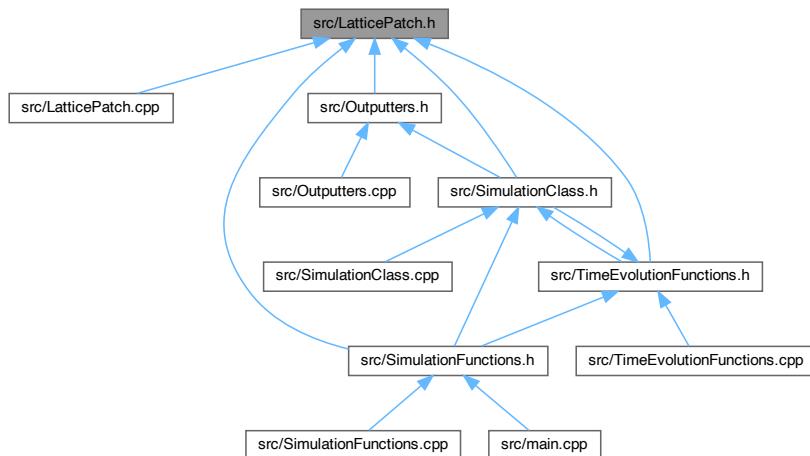
## 6.12 src/LatticePatch.h File Reference

Declaration of the lattice and lattice patches.

```
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <array>
#include <vector>
#include <algorithm>
#include <mpi.h>
#include <cvode/cvode.h>
#include <nvector/nvector_parallel.h>
#include <sundials/sundials_types.h>
#include "DerivationStencils.h"
Include dependency graph for LatticePatch.h:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [Lattice](#)  
`Lattice` class for the construction of the enveloping discrete simulation space.
- class [LatticePatch](#)  
`LatticePatch` class for the construction of the patches in the enveloping lattice.

## Functions

- void [errorKill](#) (const std::string &errorMessage)

- helper function for error messages
- int **check\_error** (int error, const char \*funcname, int id)
  - helper function to check MPI errors
- int **check\_retval** (void \*returnvalue, const char \*funcname, int opt, int id)
  - helper function to check CVode errors

## Variables

- constexpr unsigned int **FLatticeDimensionSet** = 0x01
  - lattice construction checking flag*
- constexpr unsigned int **FLatticePatchSetUp** = 0x01
- constexpr unsigned int **TranslocationLookupSetUp** = 0x02
- constexpr unsigned int **GhostLayersInitialized** = 0x04
- constexpr unsigned int **BuffersInitialized** = 0x08

### 6.12.1 Detailed Description

Declaration of the lattice and lattice patches.

Definition in file [LatticePatch.h](#).

### 6.12.2 Function Documentation

#### 6.12.2.1 **check\_error()**

```
int check_error (
    int error,
    const char * funcname,
    int id )
```

helper function to check MPI errors

Check MPI errors. Error handler must be set.

Definition at line 928 of file [LatticePatch.cpp](#).

```
00928
00929     int eclass, len;
00930     char errorstring[MPI_MAX_ERROR_STRING];
00931     if( error != MPI_SUCCESS ) {
00932         MPI_Error_class(error,&eclass);
00933         MPI_Error_string(error,errorstring,&len);
00934         std::cerr << "MPI Error(process " << id << ") in " << funcname << " : "
00935             << errorstring << ", from class " << eclass << std::endl;
00936     return 1;
00937 }
00938 return 0;
00939 }
```

### 6.12.2.2 check\_retval()

```
int check_retval (
    void * returnvalue,
    const char * funcname,
    int opt,
    int id )
```

helper function to check CVode errors

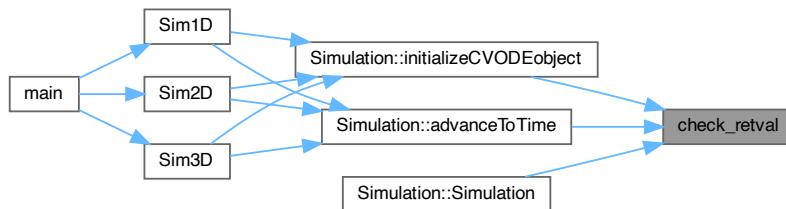
Check function return value. Adapted from CVode examples. opt == 0 means SUNDIALS function allocates memory so check if returned NULL pointer opt == 1 means SUNDIALS function returns an integer value so check if retval < 0 opt == 2 means function allocates memory so check if returned NULL pointer

Definition at line 948 of file [LatticePatch.cpp](#).

```
00948
00949     int *retval = nullptr;
00950
00951     /* Check if SUNDIALS function returned NULL pointer - no memory allocated */
00952     if (opt == 0 && returnvalue == nullptr) {
00953         fprintf(stderr,
00954             "\nSUNDIALS_ERROR(%d): %s() failed - returned NULL pointer\n\n", id,
00955             funcname);
00956         return (1);
00957     }
00958
00959     /* Check if retval < 0 */
00960     else if (opt == 1) {
00961         retval = (int *)returnvalue;
00962         char *flagname = CVodeGetReturnFlagName(*retval);
00963         if (*retval < 0) {
00964             fprintf(stderr, "\nSUNDIALS_ERROR(%d): %s() failed with retval = %d: "
00965                 "%s\n\n",
00966                 id, funcname, *retval, flagname);
00967             return (1);
00968         }
00969     }
00970
00971     /* Check if function returned NULL pointer - no memory allocated */
00972     else if (opt == 2 && returnvalue == nullptr) {
00973         fprintf(stderr,
00974             "\nMEMORY_ERROR(%d): %s() failed - returned NULL pointer\n\n", id,
00975             funcname);
00976         return (1);
00977     }
00978
00979     return (0);
00980 }
```

Referenced by [Simulation::advanceToTime\(\)](#), [Simulation::initializeCVODEobject\(\)](#), and [Simulation::Simulation\(\)](#).

Here is the caller graph for this function:



### 6.12.2.3 errorKill()

```
void errorKill (
    const std::string & errorMessage )
```

helper function for error messages

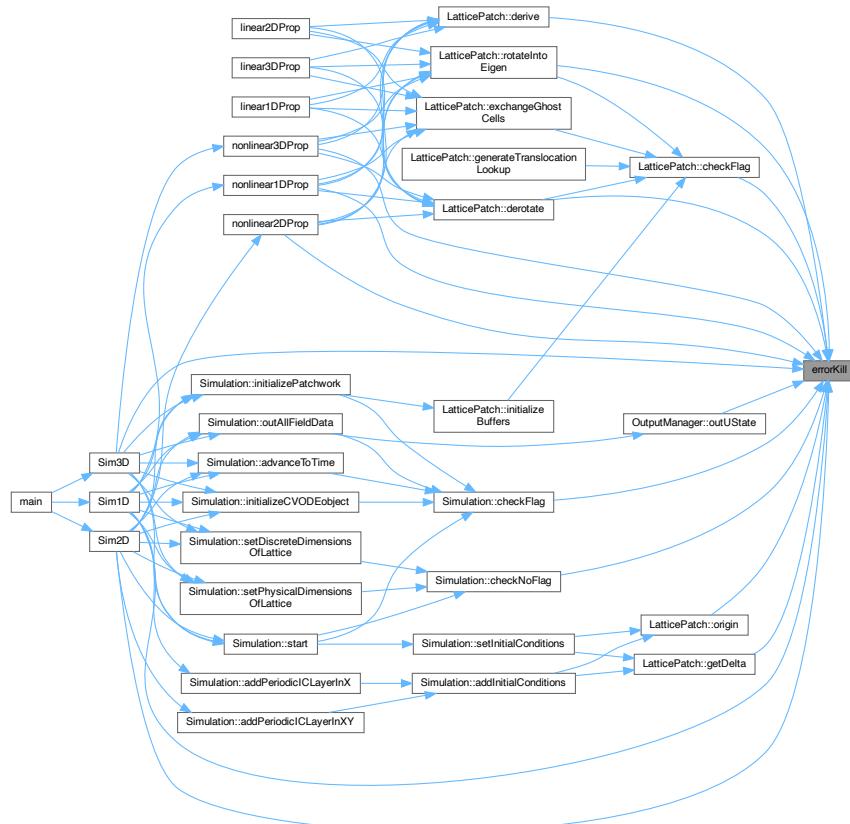
helper function for error messages

Definition at line 916 of file [LatticePatch.cpp](#).

```
00916
00917     int my_prc=0;
00918     MPI_Comm_rank(MPI_COMM_WORLD,&my_prc);
00919     if (my_prc==0) {
00920         std::cerr << std::endl << "Error: " << errorMessage
00921         << "\nAborting..." << std::endl;
00922         MPI_Abort(MPI_COMM_WORLD, 1);
00923         return;
00924     }
00925 }
```

Referenced by [LatticePatch::checkFlag\(\)](#), [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::getDelta\(\)](#), [nonlinear1DProp\(\)](#), [nonlinear2DProp\(\)](#), [nonlinear3DProp\(\)](#), [LatticePatch::origin\(\)](#), [OutputManager::outUState\(\)](#), [LatticePatch::rotateIntoEigen\(\)](#), [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



### 6.12.3 Variable Documentation

### 6.12.3.1 **BuffersInitialized**

```
constexpr unsigned int BuffersInitialized = 0x08 [constexpr]
```

lattice patch construction checking flag

Definition at line 47 of file [LatticePatch.h](#).

Referenced by [LatticePatch::checkFlag\(\)](#), and [LatticePatch::initializeBuffers\(\)](#).

### 6.12.3.2 **FLatticeDimensionSet**

```
constexpr unsigned int FLatticeDimensionSet = 0x01 [constexpr]
```

lattice construction checking flag

Definition at line 40 of file [LatticePatch.h](#).

Referenced by [LatticePatch::exchangeGhostCells\(\)](#), [LatticePatch::generateTranslocationLookup\(\)](#), [LatticePatch::initializeBuffers\(\)](#), and [Lattice::setPhysicalDimensions\(\)](#).

### 6.12.3.3 **FLatticePatchSetUp**

```
constexpr unsigned int FLatticePatchSetUp = 0x01 [constexpr]
```

lattice patch construction checking flag

Definition at line 44 of file [LatticePatch.h](#).

Referenced by [LatticePatch::checkFlag\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), [LatticePatch::rotateIntoEigen\(\)](#), and [LatticePatch::~LatticePatch\(\)](#).

### 6.12.3.4 **GhostLayersInitialized**

```
constexpr unsigned int GhostLayersInitialized = 0x04 [constexpr]
```

lattice patch construction checking flag

Definition at line 46 of file [LatticePatch.h](#).

Referenced by [LatticePatch::checkFlag\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

### 6.12.3.5 TranslocationLookupSetUp

```
constexpr unsigned int TranslocationLookupSetUp = 0x02 [constexpr]
```

lattice patch construction checking flag

Definition at line 45 of file [LatticePatch.h](#).

Referenced by [LatticePatch::checkFlag\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::generateTranslocationLookup\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

## 6.13 LatticePatch.h

[Go to the documentation of this file.](#)

```
00001 ///////////////////////////////////////////////////////////////////
00002 /// @file LatticePatch.h
00003 /// @brief Declaration of the lattice and lattice patches
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #pragma once
00007
00008 // IO
00009 #include <iomanip>
00010 #include <iostream>
00011 #include <sstream>
00012
00013 // string, container, algorithm
00014 #include <string>
00015 // #include <string_view>
00016 #include <array>
00017 #include <vector>
00018 #include <algorithm>
00019
00020 // MPI & OpenMP
00021 #include <mpi.h>
00022 #if defined(_OPENMP)
00023 #include <omp.h>
00024 #endif
00025
00026 // Sundials
00027 #include <cvode/cvode.h> /* prototypes for CVODE fcts. */
00028 #if defined(_OPENMP)
00029 #include <nvector/nvector_openmp.h> /* definition of OpenMP N_Vector */
00030 #include <nvector/nvector_mpiplusx.h> /* definition of MPI+X N_Vector */
00031 #else
00032 #include <nvector/nvector_parallel.h> /* definition of MPI N_Vector */
00033 #endif
00034 #include <sundials/sundials_types.h> /* definition of type sunrealtype */
00035
00036 // stencils
00037 #include "DerivationStencils.h"
00038
00039 /// lattice construction checking flag
00040 constexpr unsigned int FLatticeDimensionSet = 0x01;
00041
00042 /**
00043 ** lattice patch construction checking flag */
00044 constexpr unsigned int FLatticePatchSetUp = 0x01;
00045 constexpr unsigned int TranslocationLookupSetUp = 0x02;
00046 constexpr unsigned int GhostLayersInitialized = 0x04;
00047 constexpr unsigned int BuffersInitialized = 0x08;
00048 /**
00049
00050 /** @brief Lattice class for the construction of the enveloping discrete
00051 * simulation space */
00052 class Lattice {
00053 private:
00054     // physical size of the lattice in x-direction
00055     sunrealtype tot_lx;
00056     // physical size of the lattice in y-direction
00057     sunrealtype tot_ly;
00058     // physical size of the lattice in z-direction
00059     sunrealtype tot_lz;
00060     // number of points in x-direction
00061     sunindextype tot_nx;
00062     // number of points in y-direction
```

```

00063     sunindextype tot_ny;
00064     /// number of points in z-direction
00065     sunindextype tot_nz;
00066     /// total number of lattice points
00067     sunindextype tot_noP;
00068     /// dimension of each data point set once and for all
00069     static constexpr int dataPointDimension = 6;
00070     /// number of lattice points times data dimension of each point
00071     sunindextype tot_noDP;
00072     /// physical distance between lattice points in x-direction
00073     sunrealtype dx;
00074     /// physical distance between lattice points in y-direction
00075     sunrealtype dy;
00076     /// physical distance between lattice points in z-direction
00077     sunrealtype dz;
00078     /// stencil order
00079     const int stencilOrder;
00080     /// required width of ghost layers (depends on the stencil order)
00081     const int ghostLayerWidth;
00082     /// lattice status flags
00083     unsigned int statusFlags;
00084
00085 public:
00086     /// number of MPI processes
00087     int n_prc;
00088     /// number of MPI process
00089     int my_prc;
00090     /// personal communicator of the lattice
00091     MPI_Comm comm;
00092     /// function to create and deploy the cartesian communicator
00093     void initializeCommunicator(const int Nx, const int Ny,
00094         const int Nz, const bool per);
00095     /// default construction
00096     Lattice(const int St0);
00097     /// SUNContext object
00098     SUNContext sunctx;
00099     /// component function for resizing the discrete dimensions of the lattice
00100     void setDiscreteDimensions(const sunindextype _nx,
00101         const sunindextype _ny, const sunindextype _nz);
00102     /// component function for resizing the physical size of the lattice
00103     void setPhysicalDimensions(const sunrealtype _lx,
00104         const sunrealtype _ly, const sunrealtype _lz);
00105     // /**
00106     /** @brief getter function */
00107     [[nodiscard]] const sunrealtype &get_tot_lx() const { return tot_lx; }
00108     [[nodiscard]] const sunrealtype &get_tot_ly() const { return tot_ly; }
00109     [[nodiscard]] const sunrealtype &get_tot_lz() const { return tot_lz; }
00110     [[nodiscard]] const sunindextype &get_tot_nx() const { return tot_nx; }
00111     [[nodiscard]] const sunindextype &get_tot_ny() const { return tot_ny; }
00112     [[nodiscard]] const sunindextype &get_tot_nz() const { return tot_nz; }
00113     [[nodiscard]] const sunindextype &get_tot_noP() const { return tot_noP; }
00114     [[nodiscard]] const sunrealtype &get_dx() const { return dx; }
00115     [[nodiscard]] const sunrealtype &get_dy() const { return dy; }
00116     [[nodiscard]] const sunrealtype &get_dz() const { return dz; }
00117     [[nodiscard]] constexpr int get_dataPointDimension() const {
00118         return dataPointDimension;
00119     }
00120 }
00121 [[nodiscard]] const int &get_stencilOrder() const { return stencilOrder; }
00122 [[nodiscard]] const int &get_ghostLayerWidth() const {
00123     return ghostLayerWidth;
00124 }
00125 // /**
00126 };
00127
00128 /** @brief LatticePatch class for the construction of the patches in the
00129 * enveloping lattice */
00130 class LatticePatch {
00131 private:
00132     /// origin of the patch in physical space; x-coordinate
00133     sunrealtype x0;
00134     /// origin of the patch in physical space; y-coordinate
00135     sunrealtype y0;
00136     /// origin of the patch in physical space; z-coordinate
00137     sunrealtype z0;
00138     /// inner position of lattice-patch in the lattice patchwork; x-points
00139     sunindextype LIx;
00140     /// inner position of lattice-patch in the lattice patchwork; y-points
00141     sunindextype LIy;
00142     /// inner position of lattice-patch in the lattice patchwork; z-points
00143     sunindextype LIz;
00144     /// physical size of the lattice-patch in the x-dimension
00145     sunrealtype lx;
00146     /// physical size of the lattice-patch in the y-dimension
00147     sunrealtype ly;
00148     /// physical size of the lattice-patch in the z-dimension
00149     sunrealtype lz;

```

```

00150  /// number of points in the lattice patch in the x-dimension
00151  sunindextype nx;
00152  /// number of points in the lattice patch in the y-dimension
00153  sunindextype ny;
00154  /// number of points in the lattice patch in the z-dimension
00155  sunindextype nz;
00156  /// physical distance between lattice points in x-direction
00157  sunrealtype dx;
00158  /// physical distance between lattice points in y-direction
00159  sunrealtype dy;
00160  /// physical distance between lattice points in z-direction
00161  sunrealtype dz;
00162  /// lattice patch status flags
00163  unsigned int statusFlags;
00164  /// pointer to the enveloping lattice
00165  const Lattice *envelopeLattice;
00166  /// aid (auxilliarily) vector including ghost cells to compute the derivatives
00167  std::vector<sunrealtype> uAux;
00168  ///////////////////////////////////////////////////
00169  /** translocation lookup table */
00170  std::vector<sunindextype> uTox, uToy, uToz, xTou, yTou, zTou;
00171  ///////////////////////////////////////////////////
00172  ///////////////////////////////////////////////////
00173  /** buffer to save spatial derivative values */
00174  std::vector<sunrealtype> buffX, buffY, buffZ;
00175  ///////////////////////////////////////////////////
00176  ///////////////////////////////////////////////////
00177  /** buffer for passing ghost cell data */
00178  std::vector<sunrealtype> ghostCellLeft, ghostCellRight, ghostCellLeftToSend,
00179  ghostCellRightToSend, ghostCellsToSend, ghostCells;
00180  ///////////////////////////////////////////////////
00181  ///////////////////////////////////////////////////
00182  /** ghost cell translocation lookup table */
00183  std::vector<sunindextype> lgcTox, rgcTox, lgcToy, rgcToy, lgcToz, rgcToz;
00184  ///////////////////////////////////////////////////
00185  ///////////////////////////////////////////////////
00186  /** Rotate and translocate an input array according to a lookup into an output
00187  * array */
00188  inline void rotateToX(sunrealtype *outArray, const unrealtype *inArray,
00189  const std::vector<sunindextype> &lookup);
00190  inline void rotateToY(sunrealtype *outArray, const unrealtype *inArray,
00191  const std::vector<sunindextype> &lookup);
00192  inline void rotateToZ(sunrealtype *outArray, const unrealtype *inArray,
00193  const std::vector<sunindextype> &lookup);
00194  ///////////////////////////////////////////////////
00195  public:
00196  /// ID of the LatticePatch, corresponds to process number (for debugging)
00197  int ID;
00198  /// NVector for saving field components u=(E,B) in lattice points
00199  N_Vector uLocal, u;
00200  /// NVector for saving temporal derivatives of the field data
00201  N_Vector duLocal, du;
00202  /// pointer to field data
00203  unrealtype *uData;
00204  /// pointer to time-derivative data
00205  unrealtype *duData;
00206  /// pointer to auxiliary data vector
00207  unrealtype *uAuxData;
00208  ///////////////////////////////////////////////////
00209  /** pointer to halo data */
00210  unrealtype *gCLData, *gCRData;
00211  ///////////////////////////////////////////////////
00212  /** pointer to spatial derivative data buffers
00213  std::array<sunrealtype *, 3> buffData;
00214  /// constructor setting up a default first lattice patch
00215  LatticePatch();
00216  /// destructor freeing parallel vectors
00217  ~LatticePatch();
00218  /// friend function for creating the patchwork slicing of the overall lattice
00219  friend int generatePatchwork(const Lattice &envelopeLattice,
00220  LatticePatch &patchToMold, const int DLx,
00221  const int DLy, const int DLz);
00222  /// function to get the discrete size of the LatticePatch
00223  sunindextype discreteSize(int dir=0) const;
00224  /// function to get the origin of the patch
00225  unrealtype origin(const int dir) const;
00226  /// function to get distance between points
00227  unrealtype getDelta(const int dir) const;
00228  /// function to fill out the lookup tables for cache efficiency
00229  void generateTranslocationLookup();
00230  /// function to rotate u into Z-matrix eigenraum
00231  void rotateIntoEigen(const int dir);
00232  /// function to derotate uAux into dudata lattice direction of x
00233  void derotate(int dir, unrealtype *buffOut);
00234  /// initialize buffers to save derivatives
00235  void initializeBuffers();
00236  /// function to exchange ghost cells

```

```

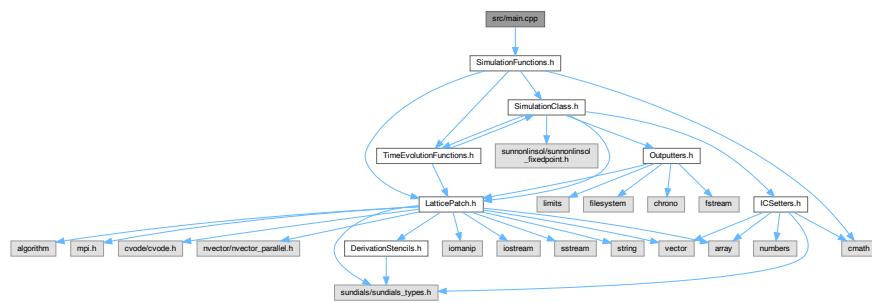
00237 void exchangeGhostCells(const int dir);
00238 /// function to derive the centered values in uAux and save them noncentered
00239 void derive(const int dir);
00240 /// function to check if a flag has been set and if not abort
00241 void checkFlag(unsigned int flag) const;
00242 };
00243
00244 /// helper function for error messages
00245 void errorKill(const std::string & errorMessage);
00246
00247 /// helper function to check MPI errors
00248 int check_error(int error, const char *funcname, int id);
00249
00250 /// helper function to check CVode errors
00251 int check_retval(void *returnvalue, const char *funcname, int opt, int id);
00252

```

## 6.14 src/main.cpp File Reference

Main function to configure the user's simulation settings.

```
#include "SimulationFunctions.h"
Include dependency graph for main.cpp:
```



## Functions

- int `main` (int argc, char \*argv[ ])

### 6.14.1 Detailed Description

Main function to configure the user's simulation settings.

Definition in file [main.cpp](#).

### 6.14.2 Function Documentation

### 6.14.2.1 main()

```
int main (
    int argc,
    char * argv[ ] )
```

Determine the output directory.

A "SimResults" folder will be created if non-existent with a subdirectory named in the identifier format "yy-mm-dd\_hh-MM-ss" that contains the csv files

A 1D simulation with specified

- relative and absolute tolerances of the CVode solver
- accuracy order of the stencils in the range 1-13
- physical length of the lattice in meters
- number of lattice points
- periodic or vanishing boundary values
- included processes of the weak-field expansion, see [README.md](#)
- physical total simulation time
- discrete time steps
- output step multiples
- output in csv (c) or binary (b) format

Add electromagnetic waves.

A plane wave with

- wavevector (normalized to  $1/\lambda$ )
- amplitude/polarization
- phase shift

Another plane wave with

- wavevector (normalized to  $1/\lambda$ )
- amplitude/polarization
- phase shift

A Gaussian wave with

- wavevector (normalized to  $1/\lambda$ )
- polarization/amplitude
- shift from origin

- width
- phase shift

Another Gaussian with

- wavevector (normalized to  $1/\lambda$ )
- polarization/amplitude
- shift from origin
- width
- phase shift

A 2D simulation with specified

- relative and absolute tolerances of the CVode solver
- accuracy order of the stencils in the range 1-13
- physical length of the lattice in the given dimensions in meters
- number of lattice points per dimension
- slicing of discrete dimensions into patches
- periodic or vanishing boundary values
- included processes of the weak-field expansion, see [README.md](#)
- physical total simulation time
- discrete time steps
- output step multiples
- output in csv (c) or binary (b) format

Add electromagnetic waves.

A plane wave with

- wavevector (normalized to  $1/\lambda$ )
- amplitude/polarization
- phase shift

Another plane wave with

- wavevector
- amplitude/polarization
- phase shift

A Gaussian wave with

- center it approaches
- normalized direction *from* which the wave approaches the center
- amplitude
- polarization rotation from TE-mode (z-axis)
- taille
- Rayleigh length

the wavelength is determined by the relation  $\lambda = \pi * w_0^2 / z_R$

- beam center
- beam length

Another Gaussian wave with

- center it approaches
- normalized direction from which the wave approaches the center
- amplitude
- polarization rotation fom TE-mode (z-axis)
- taille
- Rayleigh length
- beam center
- beam length

A 3D simulation with specified

- relative and absolute tolerances of the CVode solver
- accuracy order of the stencils in the range 1-13
- physical dimensions in meters
- number of lattice points in any dimension
- slicing of discrete dimensions into patches
- perodic or non-periodic boundaries
- processes of the weak-field expansion, see [README.md](#)
- physical total simulation time
- discrete time steps
- output step multiples
- output in csv (c) or binary (b) format

Add electromagnetic waves.

A plane wave with

- wavevector (normalized to  $1/\lambda$ )
- amplitude/polarization
- phase shift

Another plane wave with

- wavevector (normalized to  $1/\lambda$ )
- amplitude/polarization
- phase shift

A Gaussian wave with

- center it approaches
- normalized direction *from* which the wave approaches the center
- amplitude
- polarization rotation from TE-mode (z-axis)
- taille
- Rayleigh length

the wavelength is determined by the relation  $\lambda = \pi * w_0^2 / z_R$

- beam center
- beam length

Another Gaussian wave with

- center it approaches
- normalized direction from which the wave approaches the center
- amplitude
- polarization rotation from TE-mode (z-axis)
- taille
- Rayleigh length
- beam center
- beam length

Definition at line 9 of file main.cpp.

```

00010 {
00011     /** Determine the output directory.
00012      * A "SimResults" folder will be created if non-existent
00013      * with a subdirectory named in the identifier format
00014      * "yy-mm-dd_hh-MM-ss" that contains the csv files      */
00015     constexpr auto outputDirectory = "/path/to/directory/";
00016
00017     if(!filesystem::exists(outputDirectory)) {
00018         cerr<<"\nOutput directory nonexistent.\n";
00019         exit(1);
00020     }
00021
00022     // Initialize MPI environment
00023     int provided;
00024     MPI_Init_thread(&argc, &argv, MPI_THREAD_SINGLE, &provided);
00025
00026
00027     //----- BEGIN OF CONFIGURATION -----//
00028
00029     ////////////////// -- 1D -- /////////////////////
00030     /** A 1D simulation with specified */
00031
00032     /// Specify your settings here ///
00033     constexpr array <sunrealtype,2> CVodeTolerances={1.0e-16,1.0e-16}; // - relative and absolute
00034     tolerances of the CVode solver                                         // - accuracy order of the
00035     constexpr int StencilOrder=13;                                         // - stencils in the range 1-13
00036     constexpr unrealtype physical_sidelength=300e-6;                      // - physical length of the
00037     lattice in meters                                                 // - number of lattice points
00038     constexpr sunindextype latticepoints=6e3;                            // - periodic or vanishing
00039     constexpr bool periodic=true;                                         // - boundary values
00040     int processOrder=3;                                                 // - included processes of the
00041     weak-field expansion, see README.md                                // - physical total
00042     constexpr unrealtype simulationTime=100.0e-61;                     // - simulation time
00043     constexpr int numberOfSteps=100;                                       // - discrete time steps
00044     constexpr int outputStep=100;                                         // - output step multiples
00045     constexpr char outputStyle='c';                                       // - output in csv (c) or
00046     binary (b) format
00047
00048     // Add electromagnetic waves.
00049     planewave plane1;                                                 // A plane wave with
00050     plane1.k = {1e5,0,0};                                              // - wavevector (normalized to \f$ 1/\lambda \f$)
00051     plane1.p = {0,0,0.1};                                             // - amplitude/polarization
00052     plane1.phi = {0,0,0};                                              // - phase shift
00053     planewave plane2;                                                 // Another plane wave with
00054     plane2.k = {-1e6,0,0};                                              // - wavevector (normalized to \f$ 1/\lambda \f$)
00055     plane2.p = {0,0,0.5};                                             // - amplitude/polarization
00056     plane2.phi = {0,0,0};                                              // - phase shift
00057     // Do not comment out this vector, even if no plane wave is used. But if, emplace used plane
00058     // waves.
00059     vector<planewave> planewaves;
00060     //planewaves.emplace_back(plane1);
00061     //planewaves.emplace_back(plane2);
00062
00063     gaussian1D gauss1;                                                 // A Gaussian wave with
00064     gauss1.k = {1.0e6,0,0};                                              // - wavevector (normalized to \f$ 1/\lambda \f$)
00065     gauss1.p = {0,0,0.1};                                              // - polarization/amplitude
00066     gauss1.x0 = {100e-6,0,0};                                           // - shift from origin
00067     gauss1.phig = 5e-6;                                                 // - width
00068     gauss1.phi = {0,0,0};                                              // - phase shift
00069     gaussian1D gauss2;                                                 // Another Gaussian with
00070     gauss2.k = {-0.2e6,0,0};                                              // - wavevector (normalized to \f$ 1/\lambda \f$)
00071     gauss2.p = {0,0,0.5};                                              // - polarization/amplitude
00072     gauss2.x0 = {200e-6,0,0};                                           // - shift from origin
00073     gauss2.phig = 15e-6;                                                // - width
00074     gauss2.phi = {0,0,0};                                              // - phase shift
00075     // Do not comment out this vector, even if no Gaussian wave is used. But if, emplace used Gaussian
00076     // waves.
00077     vector<gaussian1D> Gaussians1D;
00078     Gaussians1D.emplace_back(gauss1);
00079     Gaussians1D.emplace_back(gauss2);
00080
00081     //////////////////////////////////////////////////////////////////
00082
00083
00084
00085     ////////////////// -- 2D -- /////////////////////
00086     /** A 2D simulation with specified */

```

```

00087     ///////////////////////////////////////////////////////////////////
00088     constexpr array<sunrealtype,2> CVodeTolerances={1.0e-12,1.0e-12};      /// - relative and absolute
00089     tolerances of the CVode solver
00090     constexpr int StencilOrder=13;                                         /// - accuracy order of the
00091     stencils in the range 1-13
00092     constexpr array<sunrealtype,2> physical_sidelengths={80e-6,80e-6};    /// - physical length of the
00093     lattice in the given dimensions in meters
00094     constexpr array<sunindextype,2> latticepoints_per_dim={800,800};    /// - number of lattice points
00095     per dimension
00096     constexpr array<int,2> patches_per_dim={2,2};                /// - slicing of discrete
00097     dimensions into patches
00098     constexpr bool periodic=true;                                         /// - periodic or vanishing
00099     boundary values
00100    int processOrder=3;                                              /// - included processes of the
00101    weak-field expansion, see README.md
00102    constexpr sunrealtype simulationTime=40e-61;    /// - physical total simulation
00103    time
00104    constexpr int numberOfSteps=100;          /// - discrete time steps
00105    constexpr int outputStep=100;           /// - output step multiples
00106    constexpr char outputStyle='c';        /// - output in csv (c) or
00107    binary (b) format
00108
00109    // Add electromagnetic waves.
00110    planewave plane1;                                         /// A plane wave with
00111    plane1.k = {1e5,0,0};          /// - wavevector (normalized to \f$ 1/\lambda \f$)
00112    plane1.p = {0,0,1};          /// - amplitude/polarization
00113    plane1.phi = {0,0,0};          /// - phase shift
00114    planewave plane2;                                         /// Another plane wave with
00115    plane2.k = {-1e6,0,0};        /// - wavevector
00116    plane2.p = {0,0,0.5};        /// - amplitude/polarization
00117    plane2.phi = {0,0,0};          /// - phase shift
00118    // Do not comment out this vector, even if no plane wave is used. But if, emplace used plane
00119    waves.
00120    vector<planewave> planewaves;
00121    //planewaves.emplace_back(plane1);
00122    //planewaves.emplace_back(plane2);
00123
00124
00125    gaussian2D gauss1;                                         /// A Gaussian wave with
00126    gauss1.x0 = {40e-6,40e-6};          /// - center it approaches
00127    gauss1.axis = {1,0};            /// - normalized direction _from_ which the wave approaches the
00128    center
00129    gauss1.amp = 0.5;          /// - amplitude
00130    gauss1.phip = 2*atan(0);        /// - polarization rotation from TE-mode (z-axis)
00131    gauss1.w0 = 2.3e-6;          /// - taille
00132    gauss1.zr = 16.619e-6;        /// - Rayleigh length
00133    // The wavelength is determined by the relation \f$ \lambda = \pi * w_0^2 / z_R \f$
00134    gauss1.ph0 = 2e-5;          /// - beam center
00135    gauss1.phA = 0.45e-5;        /// - beam length
00136    gaussian2D gauss2;          /// Another Gaussian wave with
00137    gauss2.x0 = {40e-6,40e-6};        /// - center it approaches
00138    gauss2.axis = {-0.7071,0.7071};        /// - normalized direction from which the wave approaches the
00139    center
00140    gauss2.amp = 0.5;          /// - amplitude
00141    gauss2.phip = 2*atan(0);        /// - polarization rotation fom TE-mode (z-axis)
00142    gauss2.w0 = 2.3e-6;          /// - taille
00143    gauss2.zr = 16.619e-6;        /// - Rayleigh length
00144    gauss2.ph0 = 2e-5;          /// - beam center
00145    gauss2.phA = 0.45e-5;        /// - beam length
00146    // Do not comment out this vector, even if no Gaussian wave is used. But if, emplace used Gaussian
00147    waves.
00148    vector<gaussian2D> Gaussians2D;
00149    Gaussians2D.emplace_back(gauss1);
00150    Gaussians2D.emplace_back(gauss2);
00151
00152
00153    ///////////////////////////////////////////////////////////////// -- 3D -- ///////////////////////////////
00154    /** A 3D simulation with specified */
00155
00156    /////////////////////////////////////////////////////////////////
00157    constexpr array<sunrealtype,2> CVodeTolerances={1.0e-12,1.0e-12};      /// - relative and
00158    absolute tolerances of the CVode solver
00159    constexpr int StencilOrder=13;                                         /// - accuracy order of
00160    the stencils in the range 1-13

```

```

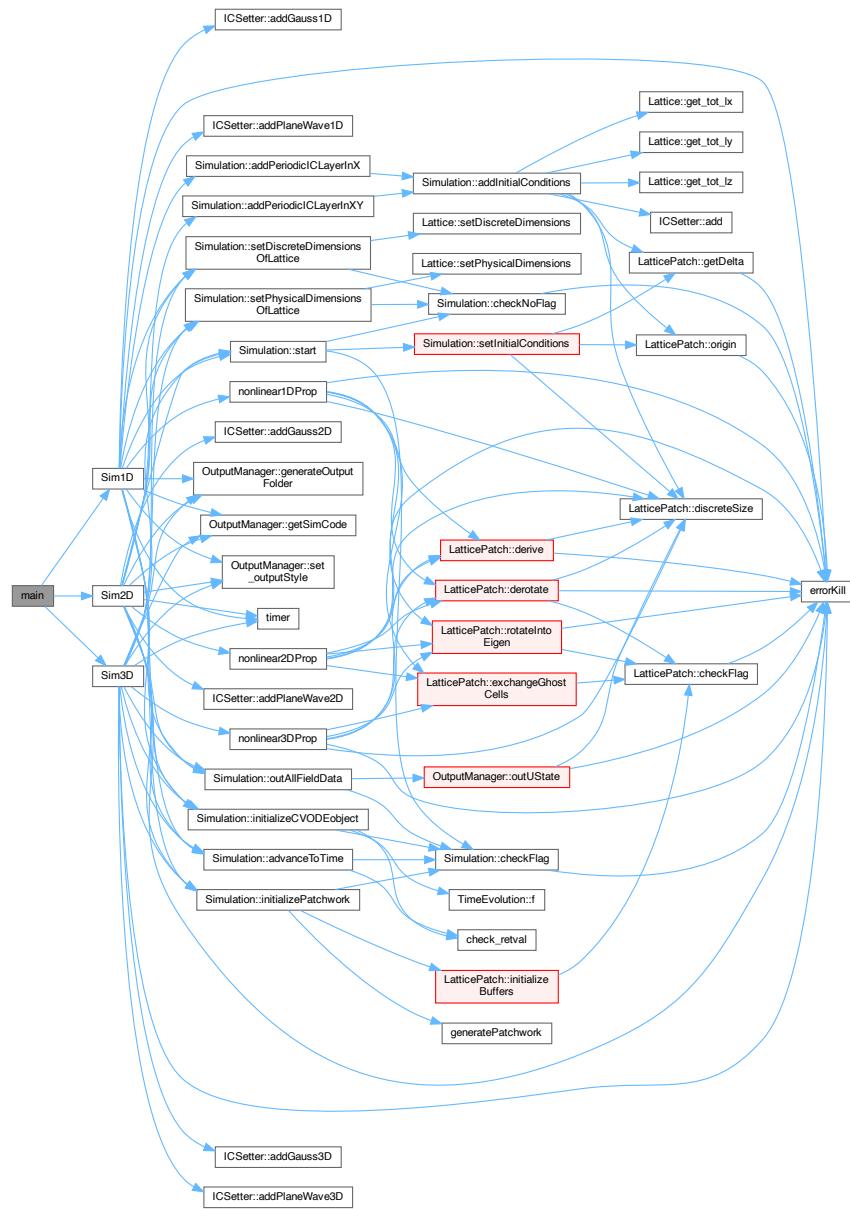
00159     constexpr arraysunrealtype,3> physical_sidelengths={80e-6,80e-6,20e-6}; /// - physical dimensions
00160     constexpr arraysunindextype,3> latticepoints_per_dim={800,800,200}; /// - number of lattice
00161     constexpr arrayint,3> patches_per_dim= {8,8,2}; /// - slicing of discrete
00162     constexpr bool periodic=true; /// - periodic or
00163     int processOrder=3; /// - processes of the
00164     constexpr sunrealtype simulationTime=40e-6; /// - physical total
00165     constexpr int numberOfSteps=40; /// - discrete time steps
00166     constexpr int outputStep=20; /// - output step
00167     constexpr char outputStyle='b'; /// - output in csv (c)
00168     or binary (b) format
00169
00170     /// Add electromagnetic waves.
00171     planewave plane1; /// A plane wave with
00172     plane1.k = {1e5,0,0}; /// - wavevector (normalized to \f$ 1/\lambda \f$)
00173     plane1.p = {0,0,0.1}; /// - amplitude/polarization
00174     plane1.phi = {0,0,0}; /// - phase shift
00175     planewave plane2; /// Another plane wave with
00176     plane2.k = {-1e6,0,0}; /// - wavevector (normalized to \f$ 1/\lambda \f$)
00177     plane2.p = {0,0,0.5}; /// - amplitude/polarization
00178     plane2.phi = {0,0,0}; /// - phase shift
00179     // Do not comment out this vector, even if no plane wave is used. But if, emplace used plane
00180     waves.
00181     vector<planewave> planewaves;
00182     //planewaves.emplace_back(plane1);
00183     //planewaves.emplace_back(plane2);
00184
00185     gaussian3D gauss1; /// A Gaussian wave with
00186     gauss1.x0 = {40e-6,40e-6,10e-6}; /// - center it approaches
00187     gauss1.axis = {1,0,0}; /// - normalized direction _from_ which the wave approaches
00188     the center
00189     gauss1.amp = 0.05; /// - amplitude
00190     gauss1.phip = 2*atan(0); /// - polarization rotation from TE-mode (z-axis)
00191     gauss1.w0 = 2.3e-6; /// - taille
00192     gauss1.zr = 16.619e-6; /// - Rayleigh length
00193     // the wavelength is determined by the relation \f$ \lambda = \pi * w_0^2 / z_R \f$
00194     gauss1.ph0 = 2e-5; /// - beam center
00195     gauss1.phA = 0.45e-5; /// - beam length
00196     gaussian3D gauss2; /// Another Gaussian wave with
00197     gauss2.x0 = {40e-6,40e-6,10e-6}; /// - center it approaches
00198     gauss2.axis = {0,1,0}; /// - normalized direction from which the wave approaches the
00199     center
00200     gauss2.amp = 0.05; /// - amplitude
00201     gauss2.phip = 2*atan(0); /// - polarization rotation from TE-mode (z-axis)
00202     gauss2.w0 = 2.3e-6; /// - taille
00203     gauss2.zr = 16.619e-6; /// - Rayleigh length
00204     gauss2.ph0 = 2e-5; /// - beam center
00205     gauss2.phA = 0.45e-5; /// - beam length
00206     // Do not comment out this vector, even if no Gaussian wave is used. But if, emplace used Gaussian
00207     waves.
00208     vector<gaussian3D> Gaussians3D;
00209     Gaussians3D.emplace_back(gauss1);
00210     Gaussians3D.emplace_back(gauss2);
00211
00212     //// Do not change this below ////
00213     static_assert(latticepoints_per_dim[0]*patches_per_dim[0]==0 &&
00214         latticepoints_per_dim[1]*patches_per_dim[1]==0 &&
00215         latticepoints_per_dim[2]*patches_per_dim[2]==0,
00216         "The number of lattice points in each dimension must be "
00217         "divisible by the number of patches in that direction.");
00218     static_assert(latticepoints_per_dim[0]/patches_per_dim[0] ==
00219         latticepoints_per_dim[1]/patches_per_dim[1] &&
00220         latticepoints_per_dim[0]/patches_per_dim[0] ==
00221         latticepoints_per_dim[2]/patches_per_dim[2],
00222         "At 3D simulations you are forced to make patches cubic in terms of "
00223         "lattice points as this is decisive for computational efficiency.");
00224     int *interactions = &processOrder;
00225     Sim3D(CVodeTolerances, StencilOrder, physical_sidelengths,
00226             latticepoints_per_dim, patches_per_dim, periodic, interactions,
00227             simulationTime, numberOfSteps, outputDirectory, outputStep,
00228             outputStyle, planewaves, Gaussians3D);
00229
00230     ////////// END OF CONFIGURATION ///////////
00231
00232     // Finalize MPI environment
00233     MPI_Finalize();
00234
00235     return 0;

```

```
00233 }
```

References `planewave::k`, `gaussian1D::k`, `planewave::p`, `gaussian1D::p`, `planewave::phi`, `gaussian1D::phi`, `gaussian1D::phig`, `Sim1D()`, `Sim2D()`, `Sim3D()`, and `gaussian1D::x0`.

Here is the call graph for this function:



## 6.15 main.cpp

[Go to the documentation of this file.](#)

```
00001 /// @file main.cpp
00002 /// @brief Main function to configure the user's simulation settings
00003
00004
00005 #include "SimulationFunctions.h" /* complete simulation functions and all headers */
```

```

00006
00007 using namespace std;
00008
00009 int main(int argc, char *argv[])
00010 {
00011     /** Determine the output directory.
00012      * A "SimResults" folder will be created if non-existent
00013      * with a subdirectory named in the identifier format
00014      * "yy-mm-dd_hh-MM-ss" that contains the csv files   */
00015     constexpr auto outputDirectory = "/path/to/directory/";
00016
00017     if(!filesystem::exists(outputDirectory)) {
00018         cerr<<"\nOutput directory nonexistent.\n";
00019         exit(1);
00020     }
00021
00022     // Initialize MPI environment
00023     int provided;
00024     MPI_Init_thread(&argc, &argv, MPI_THREAD_SINGLE, &provided);
00025
00026
00027     //----- BEGIN OF CONFIGURATION -----//
00028
00029     /////////////////// -- 1D -- ///////////////////
00030     /** A 1D simulation with specified */
00031
00032     //// Specify your settings here /////
00033     constexpr array <sunrealtype,2> CVodeTolerances={1.0e-16,1.0e-16}; // - relative and absolute
00034     tolerances of the CVode solver
00035     constexpr int StencilOrder=13;                                         // - accuracy order of the
00036     stencils in the range 1-13
00037     constexpr unrealtype physical_sidelength=300e-6;                   // - physical length of the
00038     lattice in meters
00039     constexpr sunindextype latticepoints=6e3;                            // - number of lattice points
00040     constexpr bool periodic=true;                                         // - periodic or vanishing
00041     boundary values
00042     int processOrder=3;                                                 // - included processes of the
00043     weak-field expansion, see README.md
00044     constexpr unrealtype simulationTime=100.0e-61;                      // - physical total
00045     simulation time
00046     constexpr int numberOfSteps=100;                                       // - discrete time steps
00047     constexpr int outputStep=100;                                         // - output step multiples
00048     constexpr char outputStyle='c';                                       // - output in csv (c) or
00049     binary (b) format
00050
00051     /// Add electromagnetic waves.
00052     planewave plane1;                                                 // - A plane wave with
00053     plane1.k = {1e5,0,0};                                              // - wavevector (normalized to  $1/\lambda$ )
00054     plane1.p = {0,0,0.1};                                              // - amplitude/polarization
00055     plane1.phi = {0,0,0};                                              // - phase shift
00056     planewave plane2;                                                 // - Another plane wave with
00057     plane2.k = {-1e6,0,0};                                              // - wavevector (normalized to  $1/\lambda$ )
00058     plane2.p = {0,0,0.5};                                              // - amplitude/polarization
00059     plane2.phi = {0,0,0};                                              // - phase shift
00060     // Do not comment out this vector, even if no plane wave is used. But if, emplace used plane
00061     // waves.
00062     vector<planewave> planewaves;
00063     //planewaves.emplace_back(plane1);
00064     //planewaves.emplace_back(plane2);
00065
00066     gaussian1D gauss1;                                                 // - A Gaussian wave with
00067     gauss1.k = {1.0e6,0,0};                                              // - wavevector (normalized to  $1/\lambda$ )
00068     gauss1.p = {0,0,0.1};                                              // - polarization/amplitude
00069     gauss1.x0 = {100e-6,0,0};                                            // - shift from origin
00070     gauss1.phig = 5e-6;                                                 // - width
00071     gauss1.phi = {0,0,0};                                              // - phase shift
00072     gaussian1D gauss2;                                                 // - Another Gaussian with
00073     gauss2.k = {-0.2e6,0,0};                                              // - wavevector (normalized to  $1/\lambda$ )
00074     gauss2.p = {0,0,0.5};                                              // - polarization/amplitude
00075     gauss2.x0 = {200e-6,0,0};                                            // - shift from origin
00076     gauss2.phig = 15e-6;                                                 // - width
00077     gauss2.phi = {0,0,0};                                              // - phase shift
00078     // Do not comment out this vector, even if no Gaussian wave is used. But if, emplace used Gaussian
00079     // waves.
00080     vector<gaussian1D> Gaussians1D;
00081     Gaussians1D.emplace_back(gauss1);
00082     Gaussians1D.emplace_back(gauss2);
00083
00084     //// Do not change this below /////
00085     int *interactions = &processOrder;
00086     Sim1D(CVodeTolerances,StencilOrder,physical_sidelength,latticepoints,
00087            periodic,interactions,simulationTime,numberOfSteps,
00088            outputDirectory,outputStep,outputStyle,
00089            planewaves,Gaussians1D);
00090
00091     ///////////////////////////////
00092
00093

```

```

00084
00085 ////////////////////////////////////////////////////////////////// -- 2D -- //////////////////////////////////////////////////////////////////
00086 /** A 2D simulation with specified */
00087
00088 //////////////////////////////////////////////////////////////////
00089 constexpr array<sunrealtype,2> CVodeTolerances={1.0e-12,1.0e-12}; // - relative and absolute
00090 tolerances of the CVode solver
00091 constexpr int StencilOrder=13; // - accuracy order of the
00092 stencils in the range 1-13
00093 constexpr array<sunrealtype,2> physical_sidelengths={80e-6,80e-6}; // - physical length of the
00094 lattice in the given dimensions in meters
00095 constexpr array<sunindextype,2> latticepoints_per_dim={800,800}; // - number of lattice points
00096 per dimension
00097 constexpr array<int,2> patches_per_dim={2,2}; // - slicing of discrete
00098 dimensions into patches
00099 constexpr bool periodic=true; // - periodic or vanishing
00100 boundary values
00101 int processOrder=3; // - included processes of the
00102 weak-field expansion, see README.md
00103 constexpr unrealtype simulationTime=40e-61; // - physical total simulation
00104 time
00105 constexpr int numberofSteps=100; // - discrete time steps
00106 constexpr int outputStep=100; // - output step multiples
00107 constexpr char outputStyle='c'; // - output in csv (c) or
00108 binary (b) format
00109
00110 // Add electromagnetic waves.
00111 planewave plane1; // A plane wave with
00112 plane1.k = {1e5,0,0}; // - wavevector (normalized to \f$ 1/\lambda \f$)
00113 plane1.p = {0,0,0.1}; // - amplitude/polarization
00114 plane1.phi = {0,0,0}; // - phase shift
00115 planewave plane2; // Another plane wave with
00116 plane2.k = {-1e6,0,0}; // - wavevector
00117 plane2.p = {0,0,0.5}; // - amplitude/polarization
00118 plane2.phi = {0,0,0}; // - phase shift
00119 // Do not comment out this vector, even if no plane wave is used. But if, emplace used plane
00120 waves.
00121 vector<planewave> planewaves;
00122 //planewaves.emplace_back(plane1);
00123 //planewaves.emplace_back(plane2);
00124
00125 gaussian2D gauss1; // A Gaussian wave with
00126 gauss1.x0 = {40e-6,40e-6}; // - center it approaches
00127 gauss1.axis = {1,0}; // - normalized direction _from_ which the wave approaches the
00128 center
00129 gauss1.amp = 0.5; // - amplitude
00130 gauss1.phip = 2*atan(0); // - polarization rotation from TE-mode (z-axis)
00131 gauss1.w0 = 2.3e-6; // - taille
00132 gauss1.zr = 16.619e-6; // - Rayleigh length
00133 // the wavelength is determined by the relation \f$ \lambda = \pi w_0^2/z_R \f$
00134 gauss1.ph0 = 2e-5; // - beam center
00135 gauss1.phA = 0.45e-5; // - beam length
00136 gaussian2D gauss2; // Another Gaussian wave with
00137 gauss2.x0 = {40e-6,40e-6}; // - center it approaches
00138 gauss2.axis = {-0.7071,0.7071}; // - normalized direction from which the wave approaches the
00139 center
00140 gauss2.amp = 0.5; // - amplitude
00141 gauss2.phip = 2*atan(0); // - polarization rotation from TE-mode (z-axis)
00142 gauss2.w0 = 2.3e-6; // - taille
00143 gauss2.zr = 16.619e-6; // - Rayleigh length
00144 gauss2.ph0 = 2e-5; // - beam center
00145 gauss2.phA = 0.45e-5; // - beam length
00146 // Do not comment out this vector, even if no Gaussian wave is used. But if, emplace used Gaussian
00147 waves.
00148 vector<gaussian2D> Gaussians2D;
00149 Gaussians2D.emplace_back(gauss1);
00150 Gaussians2D.emplace_back(gauss2);
00151
00152 ////////////////////////////////////////////////////////////////// -- 3D -- //////////////////////////////////////////////////////////////////
00153 /** A 3D simulation with specified */
00154
00155 //////////////////////////////////////////////////////////////////
00156 constexpr array<sunrealtype,2> CVodeTolerances={1.0e-12,1.0e-12}; // - relative and

```

```

absolute tolerances of the CVode solver
00158    constexpr int StencilOrder=13;                                     /// - accuracy order of
the stencils in the range 1-13
00159    constexpr array<sunrealtype,3> physical_sidelengths={80e-6,80e-6,20e-6}; /// - physical dimensions
in meters
00160    constexpr array<sunindextype,3> latticepoints_per_dim={800,800,200};     /// - number of lattice
points in any dimension
00161    constexpr array<int,3> patches_per_dim= {8,8,2};                      /// - slicing of discrete
dimensions into patches
00162    constexpr bool periodic=true;                                         /// - periodic or
non-periodic boundaries
00163    int processOrder=3;                                                 /// - processes of the
weak-field expansion, see README.md
00164    constexpr unrealtype simulationTime=40e-6;                         /// - physical total
simulation time
00165    constexpr int numberOfSteps=40;                                       /// - discrete time steps
00166    constexpr int outputStep=20;                                         /// - output step
multiples
00167    constexpr char outputStyle='b';                                      /// - output in csv (c)
or binary (b) format
00168
00169    // Add electromagnetic waves.
00170    planewave plane1;                                              /// A plane wave with
00171    plane1.k = {1e5,0,0};                                            /// - wavevector (normalized to  $\lambda$ )
00172    plane1.p = {0,0,0.1};                                           /// - amplitude/polarization
00173    plane1.phi = {0,0,0};                                           /// - phase shift
00174    planewave plane2;                                              /// Another plane wave with
00175    plane2.k = {-1e6,0,0};                                          /// - wavevector (normalized to  $\lambda$ )
00176    plane2.p = {0,0,0.5};                                           /// - amplitude/polarization
00177    plane2.phi = {0,0,0};                                           /// - phase shift
00178    // Do not comment out this vector, even if no plane wave is used. But if, emplace used plane
waves.
00179    vector<planewave> planewaves;
00180    //planewaves.emplace_back(plane1);
00181    //planewaves.emplace_back(plane2);
00182
00183    gaussian3D gauss1;                                              /// A Gaussian wave with
00184    gauss1.x0 = {40e-6,40e-6,10e-6};                                /// - center it approaches
00185    gauss1.axis = {1,0,0};                                           /// - normalized direction _from_ which the wave approaches
the center
00186    gauss1.amp = 0.05;                                              /// - amplitude
00187    gauss1.phip = 2*atan(0);                                         /// - polarization rotation from TE-mode (z-axis)
00188    gauss1.w0 = 2.3e-6;                                             /// - taille
00189    gauss1.zr = 16.619e-6;                                           /// - Rayleigh length
00190    // the wavelength is determined by the relation  $\lambda = \pi w_0^2 / z_R$ 
00191    gauss1.ph0 = 2e-5;                                              /// - beam center
00192    gauss1.phA = 0.45e-5;                                            /// - beam length
00193    gaussian3D gauss2;                                              /// Another Gaussian wave with
00194    gauss2.x0 = {40e-6,40e-6,10e-6};                                /// - center it approaches
00195    gauss2.axis = {0,1,0};                                           /// - normalized direction from which the wave approaches the
center
00196    gauss2.amp = 0.05;                                              /// - amplitude
00197    gauss2.phip = 2*atan(0);                                         /// - polarization rotation from TE-mode (z-axis)
00198    gauss2.w0 = 2.3e-6;                                             /// - taille
00199    gauss2.zr = 16.619e-6;                                           /// - Rayleigh length
00200    gauss2.ph0 = 2e-5;                                              /// - beam center
00201    gauss2.phA = 0.45e-5;                                            /// - beam length
00202    // Do not comment out this vector, even if no Gaussian wave is used. But if, emplace used Gaussian
waves.
00203    vector<gaussian3D> Gaussians3D;
00204    Gaussians3D.emplace_back(gauss1);
00205    Gaussians3D.emplace_back(gauss2);
00206
00207    //// Do not change this below /////
00208    static_assert(latticepoints_per_dim[0]*patches_per_dim[0]==0 &&
00209        latticepoints_per_dim[1]*patches_per_dim[1]==0 &&
00210        latticepoints_per_dim[2]*patches_per_dim[2]==0,
00211        "The number of lattice points in each dimension must be "
00212        "divisible by the number of patches in that direction.");
00213    static_assert(latticepoints_per_dim[0]/patches_per_dim[0] ==
00214        latticepoints_per_dim[1]/patches_per_dim[1] &&
00215        latticepoints_per_dim[0]/patches_per_dim[0] ==
00216        latticepoints_per_dim[2]/patches_per_dim[2],
00217        "At 3D simulations you are forced to make patches cubic in terms of "
00218        "lattice points as this is decisive for computational efficiency.");
00219    int *interactions = &processOrder;
00220    Sim3D(CVodeTolerances,StencilOrder,physical_sidelengths,
00221        latticepoints_per_dim,patches_per_dim,periodic,interactions,
00222        simulationTime,numberOfSteps,outputDirectory,outputStep,
00223        outputStyle,planewaves,Gaussians3D);
00224
00225    //////////////////////////////// END OF CONFIGURATION /////////////////////
00226
00227    //----- END OF CONFIGURATION -----//
00228
00229    // Finalize MPI environment
00230    MPI_Finalize();

```

```

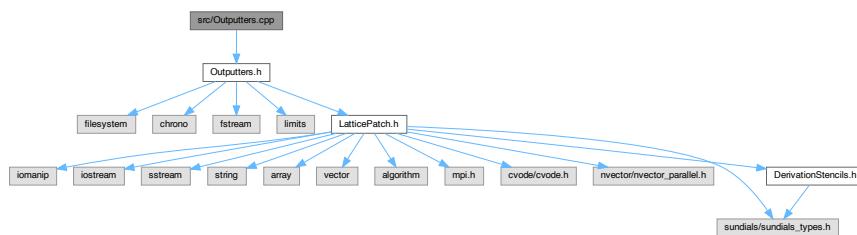
00231
00232     return 0;
00233 }
```

## 6.16 src/Outputters.cpp File Reference

Generation of output writing to disk.

```
#include "Outputters.h"
```

Include dependency graph for Outputters.cpp:



### 6.16.1 Detailed Description

Generation of output writing to disk.

Definition in file [Outputters.cpp](#).

## 6.17 Outputters.cpp

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file Outputters.cpp
00003 /// @brief Generation of output writing to disk
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #include "Outputters.h"
00007
00008 namespace fs = std::filesystem;
00009 namespace chrono = std::chrono;
00010
00011 /// Directly generate the simCode at construction
00012 OutputManager::OutputManager() {
00013     simCode = SimCodeGenerator();
00014     outputStyle = 'c';
00015 }
00016
00017 /// Generate the identifier number reverse from year to minute in the format
00018 /// yy-mm-dd-hh-MM-ss
00019 std::string OutputManager::SimCodeGenerator() {
00020     const chrono::time_point<chrono::system_clock> now{
00021         chrono::system_clock::now()};
00022     const chrono::year_month_day ymd{chrono::floor<chrono::days>(now)};
00023     const auto tod = now - chrono::floor<chrono::days>(now);
00024     const chrono::hh_mm_ss hms{tod};
00025
00026     std::stringstream temp;
00027     temp << std::setfill('0') << std::setw(2)
00028         << static_cast<int>(ymd.year() - chrono::years(2000)) << "-"
00029         << std::setfill('0') << std::setw(2)
00030         << static_cast<unsigned>(ymd.month()) << "-"
00031         << std::setfill('0') << std::setw(2)
```

```

00032     « static_cast<unsigned>(ymd.day()) « "_"
00033     « std::setfill('0') « std::setw(2) « hms.hours().count()
00034     « "-" « std::setfill('0')
00035     « std::setw(2) « hms.minutes().count() « "-"
00036     « std::setfill('0') « std::setw(2)
00037     « hms.seconds().count();
00038     //« "_" « hms.subseconds().count(); // subseconds render the filename
00039     // too large
00040     return temp.str();
00041 }
00042
00043 /** Generate the folder to save the data to by one process:
00044 * In the given directory it creates a direcory "SimResults" and a directory
00045 * with the simCode. The relevant part of the main file is written to a
00046 * "config.txt" file in that directory to log the settings. */
00047 void OutputManager::generateOutputFolder(const std::string &dir) {
00048     // Do this only once for the first process
00049     int myPrc;
00050     MPI_Comm_rank(MPI_COMM_WORLD, &myPrc);
00051     if (myPrc == 0) {
00052         if (!fs::is_directory(dir))
00053             fs::create_directory(dir);
00054         if (!fs::is_directory(dir + "/SimResults"))
00055             fs::create_directory(dir + "/SimResults");
00056         if (!fs::is_directory(dir + "/SimResults/" + simCode))
00057             fs::create_directory(dir + "/SimResults/" + simCode);
00058     }
00059     // path variable for the output generation
00060     Path = dir + "/SimResults/" + simCode + "/";
00061
00062     // Logging configurations from main.cpp
00063     std::ifstream fin("main.cpp");
00064     std::ofstream fout(Path + "config.txt");
00065     std::string line;
00066     int begin=1000;
00067     for (int i = 1; !fin.eof(); i++) {
00068         getline(fin, line);
00069         if (line.starts_with("    //----- B")) {
00070             begin=i;
00071         }
00072         if (i < begin) {
00073             continue;
00074         }
00075         fout « line « std::endl;
00076         if (line.starts_with("    //----- E")) {
00077             break;
00078         }
00079     }
00080     return;
00081 }
00082
00083 void OutputManager::set_outputStyle(const char _outputStyle){
00084     outputStyle = _outputStyle;
00085 }
00086
00087 /** Write the field data either in csv format to one file per each process
00088 * (patch) or in binary form to a single file. Files are stores inthe simCode
00089 * directory. For csv files the state (simulation step) denotes the
00090 * prefix and the suffix after an underscore is given by the process/patch
00091 * number. Binary files are simply named after the step number. */
00092 void OutputManager::outUState(const int &state, const Lattice &lattice,
00093     const LatticePatch &latticePatch) {
00094     switch(outputStyle) {
00095         case 'c': { // one csv file per process
00096             std::ofstream ofs;
00097             ofs.open(Path + std::to_string(state) + "_"
00098                     + std::to_string(lattice.my_prc) + ".csv");
00099             // Precision of sunrealtype in significant decimal digits; 15 for IEEE double
00100             ofs « std::setprecision(std::numeric_limits<sunrealtype>::digits10);
00101
00102             // Walk through each lattice point
00103             const sunindextype totalNP = latticePatch.discreteSize();
00104             for (sunindextype i = 0; i < totalNP * 6; i += 6) {
00105                 // Six columns to contain the field data: Ex,Ey,Ez,Bx,By,Bz
00106                 ofs « latticePatch.uData[i + 0] « "," « latticePatch.uData[i + 1] « ","
00107                 « latticePatch.uData[i + 2] « "," « latticePatch.uData[i + 3] « ","
00108                 « latticePatch.uData[i + 4] « "," « latticePatch.uData[i + 5]
00109                 « std::endl;
00110             }
00111             ofs.close();
00112             break;
00113         }
00114
00115         case 'b': { // a single binary file
00116             // Open the output file
00117             MPI_File fh;
00118             const std::string filename = Path+std::to_string(state);

```

```

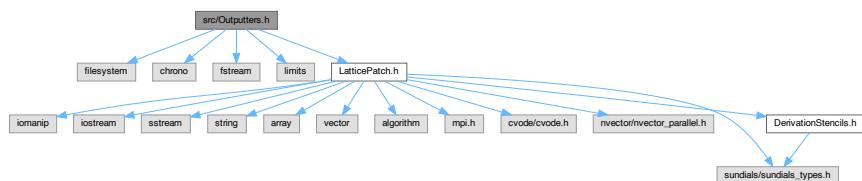
00119 MPI_File_open(lattice.comm,&filename[0],MPI_MODE_WRONLY|MPI_MODE_CREATE,
00120             MPI_INFO_NULL);
00121 // number of datapoints in the patch with process offset
00122 const sunindextype count = latticePatch.discreteSize()*
00123     lattice.get_dataPointDimension();
00124 MPI_Offset offset = lattice.my_prc*count*sizeof(MPI_SUNREALTYPE);
00125 // Go to offset in file and write data to it; maximal precision in
00126 // "native" representation
00127 MPI_File_set_view(fh,offset,MPI_SUNREALTYPE,MPI_SUNREALTYPE,"native",
00128                     MPI_INFO_NULL);
00129 MPI_Request write_request;
00130 MPI_File_iwrite_all(fh,latticePatch.uData,count,MPI_SUNREALTYPE,
00131                     &write_request);
00132 MPI_Wait(&write_request,MPI_STATUS_IGNORE);
00133 MPI_File_close(&fh);
00134 break;
00135 }
00136 default: {
00137     errorKill("No valid output style defined."
00138               " Choose between (c): one csv file per process,"
00139               " (b) one binary file");
00140     break;
00141 }
00142 }
00143 }
00144

```

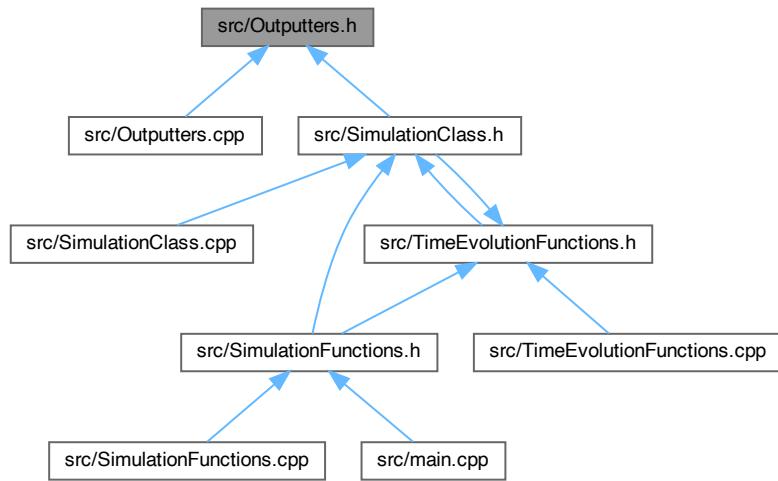
## 6.18 src/Outputters.h File Reference

[OutputManager](#) class to outstream simulation data.

```
#include <filesystem>
#include <chrono>
#include <fstream>
#include <limits>
#include "LatticePatch.h"
Include dependency graph for Outputters.h:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [OutputManager](#)  
*Output Manager class to generate and coordinate output writing to disk.*

### 6.18.1 Detailed Description

[OutputManager](#) class to outstream simulation data.

Definition in file [Outputters.h](#).

## 6.19 Outputters.h

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file Outputters.h
00003 /// @brief OutputManager class to outstream simulation data
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #pragma once
00007
00008 // perform operations on the filesystem
00009 #include <filesystem>
00010
00011 // output controlling with limits and timestep
00012 #include <chrono>
00013 #include <fstream>
00014 #include <limits>
00015
00016 // project subfile header
00017 #include "LatticePatch.h"
00018
00019 /** @brief Output Manager class to generate and coordinate output writing to
00020 * disk */
00021 class OutputManager {
00022 private:
  
```

```

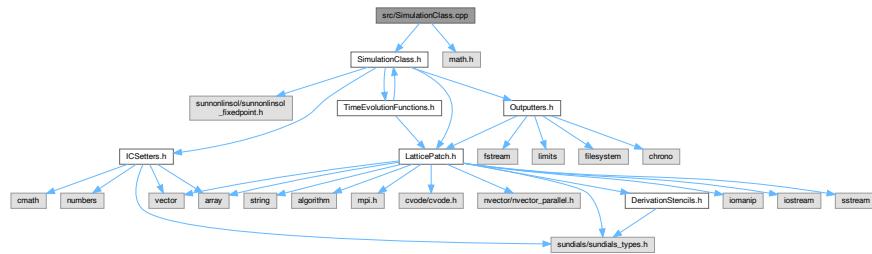
00023  /// function to create the Code of the Simulations
00024  static std::string SimCodeGenerator();
00025  /// varible to save the SimCode generated at execution
00026  std::string simCode;
00027  /// variable for the path to the output folder
00028  std::string Path;
00029  /// output style; csv or binary
00030  char outputStyle;
00031 public:
00032  /// default constructor
00033  OutputManager();
00034  /// function that creates folder to save simulation data
00035  void generateOutputFolder(const std::string &dir);
00036  /// set the output style
00037  void set_outputStyle(const char _outputStyle);
00038  /// function to write data to disk in specified way
00039  void outUState(const int &state, const Lattice &lattice,
00040      const LatticePatch &latticePatch);
00041  /// simCode getter function
00042  [[nodiscard]] const std::string &getSimCode() const { return simCode; }
00043 };
00044

```

## 6.20 src/SimulationClass.cpp File Reference

Interface to the whole [Simulation](#) procedure: from wave settings over lattice construction, time evolution and outputs (also all relevant CVODE steps are performed here)

```
#include "SimulationClass.h"
#include <math.h>
Include dependency graph for SimulationClass.cpp:
```



### 6.20.1 Detailed Description

Interface to the whole [Simulation](#) procedure: from wave settings over lattice construction, time evolution and outputs (also all relevant CVODE steps are performed here)

Definition in file [SimulationClass.cpp](#).

## 6.21 SimulationClass.cpp

[Go to the documentation of this file.](#)

```

00001  ///////////////////////////////////////////////////////////////////
00002  /// @file SimulationClass.cpp
00003  /// @brief Interface to the whole Simulation procedure:
00004  /// from wave settings over lattice construction, time evolution and outputs
00005  /// (also all relevant CVODE steps are performed here)
00006  ///////////////////////////////////////////////////////////////////
00007

```

```

00008 #include "SimulationClass.h"
00009
0010 #include <math.h>
0011
0012 /// Along with the simulation object, create the cartesian communicator and
0013 /// SUNContext object
0014 Simulation::Simulation(const int Nx, const int Ny, const int Nz,
0015     const int StencilOrder, const bool periodicity) :
0016     lattice(StencilOrder){
0017     statusFlags = 0;
0018     t = 0;
0019     // Initialize the cartesian communicator
0020     lattice.initializeCommunicator(Nx, Ny, Nz, periodicity);
0021
0022     // Create the SUNContext object associated with the thread of execution
0023     int retval = 0;
0024     retval = SUNContext_Create(&lattice.comm, &lattice.sunctx);
0025     if (check_retval(&retval, "SUNContext_Create", 1, lattice.my_prc))
0026         MPI_Abort(lattice.comm, 1);
0027 }
0028
0029 /// Free the CVode solver memory and Sundials context object with the finish of
0030 /// the simulation
0031 Simulation::~Simulation() {
0032     // Free solver memory
0033     if (statusFlags & CvodeObjectSetUp) {
0034         CVodeFree(&cvode_mem);
0035         SUNNonlinSolFree(NLS);
0036         SUNContext_Free(&lattice.sunctx);
0037     }
0038 }
0039
0040 /// Set the discrete dimensions, the number of points per dimension
0041 void Simulation::setDiscreteDimensionsOfLattice(const sunindextype nx,
0042     const sunindextype ny, const sunindextype nz) {
0043     checkNoFlag(LatticePatchworkSetUp);
0044     lattice.setDiscreteDimensions(nx, ny, nz);
0045     statusFlags |= LatticeDiscreteSetUp;
0046 }
0047
0048 /// Set the physical dimensions with lenghts in micro meters
0049 void Simulation::setPhysicalDimensionsOfLattice(const sunrealtype lx,
0050     const sunrealtype ly, const sunrealtype lz) {
0051     checkNoFlag(LatticePatchworkSetUp);
0052     lattice.setPhysicalDimensions(lx, ly, lz);
0053     statusFlags |= LatticePhysicalSetUp;
0054 }
0055
0056 /// Check that the lattice dimensions are set up and generate the patchwork
0057 void Simulation::initializePatchwork(const int nx, const int ny,
0058     const int nz) {
0059     checkFlag(LatticeDiscreteSetUp);
0060     checkFlag(LatticePhysicalSetUp);
0061
0062     // Generate the patchwork
0063     generatePatchwork(lattice, latticePatch, nx, ny, nz);
0064     latticePatch.initializeBuffers();
0065
0066     statusFlags |= LatticePatchworkSetUp;
0067 }
0068
0069 /// Configure CVODE
0070 void Simulation::initializeCVODEobject(const sunrealtype reltol,
0071     const sunrealtype abstol) {
0072     checkFlag(SimulationStarted);
0073
0074     // CVode settings return value
0075     int retval = 0;
0076
0077     // Create CVODE object -- returns a pointer to the cvode memory structure
0078     // with Adams method (Adams-Moulton formula) solver chosen for non-stiff ODE
0079     cvode_mem = CVodeCreate(CV_ADAMS, lattice.sunctx);
0080
0081     // Specify user data and attach it to the main cvode memory block
0082     retval = CVodeSetUserData(
0083         cvode_mem,
0084         &latticePatch); // patch contains the user data as used in CVRhsFn
0085     if (check_retval(&retval, "CVodeSetUserData", 1, lattice.my_prc))
0086         MPI_Abort(lattice.comm, 1);
0087
0088     // Initialize CVODE solver
0089     retval = CVodeInit(cvode_mem, TimeEvolution::f, 0,
0090         latticePatch.u); // allocate memory, CVRhsFn f, t_i=0, u
0091                                     // contains the initial values
0092     if (check_retval(&retval, "CVodeInit", 1, lattice.my_prc))
0093         MPI_Abort(lattice.comm, 1);
0094 }
```

```

00095 // Create fixed point nonlinear solver object (suitable for non-stiff ODE) and
00096 // attach it to CVode
00097 NLS = SUNNonlinSol_FixedPoint(latticePatch.u, 0, lattice.sunctx);
00098 retval = CVodeSetNonlinearSolver(cvode_mem, NLS);
00099 if (check_retval(&retval, "CVodeSetNonlinearSolver", 1, lattice.my_prc))
00100     MPI_Abort(lattice.comm, 1);
00101
00102 // Anderson damping factor
00103 retval = SUNNonlinSolSetDamping_FixedPoint(NLS,1);
00104 if (check_retval(&retval, "SUNNonlinSolSetDamping_FixedPoint", 1,
00105     lattice.my_prc)) MPI_Abort(lattice.comm, 1);
00106
00107 // Specify integration tolerances -- a scalar relative tolerance and scalar
00108 // absolute tolerance
00109 retval = CVodeSStolerances(cvode_mem, reltol, abstol);
00110 if (check_retval(&retval, "CVodeSStolerances", 1, lattice.my_prc))
00111     MPI_Abort(lattice.comm, 1);
00112
00113 // Specify the maximum number of steps to be taken by the solver in its
00114 // attempt to reach the next tout
00115 retval = CVodeSetMaxNumSteps(cvode_mem, 10000);
00116 if (check_retval(&retval, "CVodeSetMaxNumSteps", 1, lattice.my_prc))
00117     MPI_Abort(lattice.comm, 1);
00118
00119 // maximum number of warnings for too small h
00120 retval = CVodeSetMaxHnilWarns(cvode_mem,3);
00121 if (check_retval(&retval, "CVodeSetMaxHnilWarns", 1, lattice.my_prc))
00122     MPI_Abort(lattice.comm, 1);
00123
00124 statusFlags |= CvodeObjectSetUp;
00125 }
00126
00127 /// Check if the lattice patchwork is set up and set the initial conditions
00128 void Simulation::start() {
00129     checkFlag(LatticeDiscreteSetUp);
00130     checkFlag(LatticePhysicalSetUp);
00131     checkFlag(LatticePatchworkSetUp);
00132     checkNoFlag(SimulationStarted);
00133     checkNoFlag(CvodeObjectSetUp);
00134     setInitialConditions();
00135     statusFlags |= SimulationStarted;
00136 }
00137
00138 /// Set initial conditions: Fill the lattice points with the initial field
00139 /// values
00140 void Simulation::setInitialConditions() {
00141     const sunrealtyp dx = latticePatch.getDelta(1);
00142     const sunrealtyp dy = latticePatch.getDelta(2);
00143     const sunrealtyp dz = latticePatch.getDelta(3);
00144     const sunindextyp nx = latticePatch.discreteSize(1);
00145     const sunindextyp ny = latticePatch.discreteSize(2);
00146     const sunindextyp totalNP = latticePatch.discreteSize();
00147     const sunrealtyp x0 = latticePatch.origin(1);
00148     const sunrealtyp y0 = latticePatch.origin(2);
00149     const sunrealtyp z0 = latticePatch.origin(3);
00150     sunindextyp px = 0, py = 0, pz = 0;
00151     #pragma omp parallel for default(none) \
00152     shared(nx, ny, totalNP, dx, dy, dz, x0, y0, z0) \
00153     firstprivate(px, py, pz) schedule(static)
00154     for (sunindextyp i = 0; i < totalNP * 6; i += 6) {
00155         px = (i / 6) % nx;
00156         py = ((i / 6) / nx) % ny;
00157         pz = ((i / 6) / nx) / ny;
00158         // Call the 'eval' function to fill the lattice points with the field data
00159         icsettings.eval(static_cast<sunrealtyp>(px) * dx + x0,
00160                         static_cast<sunrealtyp>(py) * dy + y0,
00161                         static_cast<sunrealtyp>(pz) * dz + z0, &latticePatch.uData[i]);
00162     }
00163     return;
00164 }
00165
00166 /// Use parameters to add periodic IC layers
00167 void Simulation::addInitialConditions(const sunindextyp xm,
00168                                         const sunindextyp ym,
00169                                         const sunindextyp zm /* zm=0 always */ ) {
00170     const sunrealtyp dx = latticePatch.getDelta(1);
00171     const sunrealtyp dy = latticePatch.getDelta(2);
00172     const sunrealtyp dz = latticePatch.getDelta(3);
00173     const sunindextyp nx = latticePatch.discreteSize(1);
00174     const sunindextyp ny = latticePatch.discreteSize(2);
00175     const sunindextyp totalNP = latticePatch.discreteSize();
00176     // Correct for demanded displacement, rest as for setInitialConditions
00177     const sunrealtyp x0 = latticePatch.origin(1) + xm*lattice.get_tot_lx();
00178     const sunrealtyp y0 = latticePatch.origin(2) + ym*lattice.get_tot_ly();
00179     const sunrealtyp z0 = latticePatch.origin(3) + zm*lattice.get_tot_lz();
00180     sunindextyp px = 0, py = 0, pz = 0;
00181     for (sunindextyp i = 0; i < totalNP * 6; i += 6) {

```

```

00182     px = (i / 6) % nx;
00183     py = ((i / 6) / nx) % ny;
00184     pz = ((i / 6) / nx) / ny;
00185     icsettings.add(static_cast<sunrealtype>(px) * dx + x0,
00186                     static_cast<sunrealtype>(py) * dy + y0,
00187                     static_cast<sunrealtype>(pz) * dz + z0, &latticePatch.uData[i]);
00188 }
00189 return;
00190 }
00191
00192 /// Add initial conditions in one dimension
00193 void Simulation::addPeriodicICLayerInX() {
00194     addInitialConditions(-1, 0, 0);
00195     addInitialConditions(1, 0, 0);
00196     return;
00197 }
00198
00199 /// Add initial conditions in two dimensions
00200 void Simulation::addPeriodicICLayerInXY() {
00201     addInitialConditions(-1, -1, 0);
00202     addInitialConditions(-1, 0, 0);
00203     addInitialConditions(-1, 1, 0);
00204     addInitialConditions(0, 1, 0);
00205     addInitialConditions(0, -1, 0);
00206     addInitialConditions(1, -1, 0);
00207     addInitialConditions(1, 0, 0);
00208     addInitialConditions(1, 1, 0);
00209     return;
00210 }
00211
00212 /// Advance the solution in time -> integrate the ODE over an interval t
00213 void Simulation::advanceToTime(const unrealtype &tEnd) {
00214     checkFlag(SimulationStarted);
00215     int retval = 0;
00216     retval = CVode(cvode_mem, tEnd, latticePatch.u, &t,
00217                   CV_NORMAL); // CV_NORMAL: internal steps to reach tEnd, then
00218                   // interpolate to return latticePatch.u, return time
00219                   // reached by the solver as t
00220     if (check(retval, "CVode", 1, lattice.my_prc))
00221         MPI_Abort(lattice.comm, 1);
00222 }
00223
00224 /// Write specified simulation steps to disk
00225 void Simulation::outAllFieldData(const int &state) {
00226     checkFlag(SimulationStarted);
00227     outputManager.outUState(state, lattice, latticePatch);
00228 }
00229
00230 /// Check presence of configuration flags
00231 void Simulation::checkFlag(unsigned int flag) const {
00232     if (!(statusFlags & flag)) {
00233         std::string errorMessage;
00234         switch (flag) {
00235             case LatticeDiscreteSetUp:
00236                 errorMessage = "The discrete size of the Simulation has not been set up";
00237                 break;
00238             case LatticePhysicalSetUp:
00239                 errorMessage = "The physical size of the Simulation has not been set up";
00240                 break;
00241             case LatticePatchworkSetUp:
00242                 errorMessage = "The patchwork for the Simulation has not been set up";
00243                 break;
00244             case CvodeObjectSetUp:
00245                 errorMessage = "The CVODE object has not been initialized";
00246                 break;
00247             case SimulationStarted:
00248                 errorMessage = "The Simulation has not been started";
00249                 break;
00250             default:
00251                 errorMessage = "Uppss, you've made a non-standard error, sadly I can't "
00252                             "help you there";
00253                 break;
00254         }
00255         errorKill(errorMessage);
00256     }
00257     return;
00258 }
00259
00260 /// Check absence of configuration flags
00261 void Simulation::checkNoFlag(unsigned int flag) const {
00262     if ((statusFlags & flag)) {
00263         std::string errorMessage;
00264         switch (flag) {
00265             case LatticeDiscreteSetUp:
00266                 errorMessage =
00267                     "The discrete size of the Simulation has already been set up";
00268                 break;

```

```

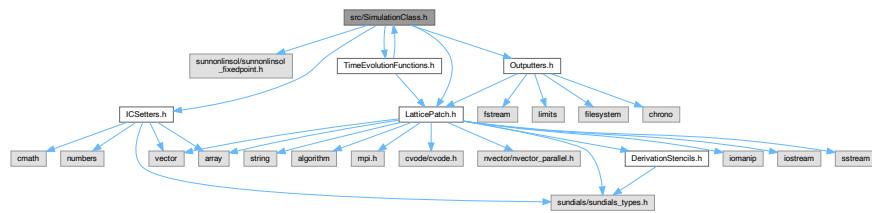
00269     case LatticePhysicalSetUp:
00270         errorMessage =
00271             "The physical size of the Simulation has already been set up";
00272         break;
00273     case LatticePatchworkSetUp:
00274         errorMessage = "The patchwork for the Simulation has already been set up";
00275         break;
00276     case CvodeObjectSetUp:
00277         errorMessage = "The CVODE object has already been initialized";
00278         break;
00279     case SimulationStarted:
00280         errorMessage = "The simulation has already started, some changes are no "
00281             "longer possible";
00282         break;
00283     default:
00284         errorMessage = "Uppss, you've made a non-standard error, sadly I can't "
00285             "help you there";
00286         break;
00287     }
00288     errorKill(errorMessage);
00289 }
00290 return;
00291 }
```

## 6.22 src/SimulationClass.h File Reference

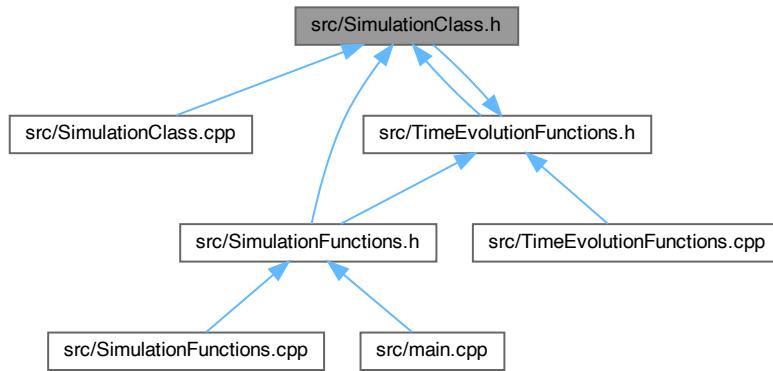
Class for the [Simulation](#) object calling all functionality: from wave settings over lattice construction, time evolution and outputs initialization of the [CVode](#) object.

```
#include "sunnonlinsol/sunnonlinsol_fixedpoint.h"
#include "ICSetters.h"
#include "LatticePatch.h"
#include "Outputters.h"
#include "TimeEvolutionFunctions.h"
```

Include dependency graph for `SimulationClass.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- class [Simulation](#)

*Simulation* class to instantiate the whole walkthrough of a [Simulation](#).

## Variables

- constexpr unsigned int [LatticeDiscreteSetUp](#) = 0x01
- constexpr unsigned int [LatticePhysicalSetUp](#) = 0x02
- constexpr unsigned int [LatticePatchworkSetUp](#) = 0x04
- constexpr unsigned int [CvodeObjectSetUp](#) = 0x08
- constexpr unsigned int [SimulationStarted](#) = 0x10

### 6.22.1 Detailed Description

Class for the [Simulation](#) object calling all functionality: from wave settings over lattice construction, time evolution and outputs initialization of the [CVode](#) object.

Definition in file [SimulationClass.h](#).

### 6.22.2 Variable Documentation

### 6.22.2.1 CvodeObjectSetUp

```
constexpr unsigned int CvodeObjectSetUp = 0x08 [constexpr]
```

simulation checking flag

Definition at line 24 of file [SimulationClass.h](#).

Referenced by [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::start\(\)](#), and [Simulation::~Simulation\(\)](#).

### 6.22.2.2 LatticeDiscreteSetUp

```
constexpr unsigned int LatticeDiscreteSetUp = 0x01 [constexpr]
```

simulation checking flag

Definition at line 21 of file [SimulationClass.h](#).

Referenced by [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [Simulation::initializePatchwork\(\)](#), [Simulation::setDiscreteDimensions\(\)](#), and [Simulation::start\(\)](#).

### 6.22.2.3 LatticePatchworkSetUp

```
constexpr unsigned int LatticePatchworkSetUp = 0x04 [constexpr]
```

simulation checking flag

Definition at line 23 of file [SimulationClass.h](#).

Referenced by [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [Simulation::initializePatchwork\(\)](#), [Simulation::setDiscreteDimensions\(\)](#), [Simulation::setPhysicalDimensionsOfLattice\(\)](#), and [Simulation::start\(\)](#).

### 6.22.2.4 LatticePhysicalSetUp

```
constexpr unsigned int LatticePhysicalSetUp = 0x02 [constexpr]
```

simulation checking flag

Definition at line 22 of file [SimulationClass.h](#).

Referenced by [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [Simulation::initializePatchwork\(\)](#), [Simulation::setPhysicalDimensions\(\)](#), and [Simulation::start\(\)](#).

### 6.22.2.5 SimulationStarted

```
constexpr unsigned int SimulationStarted = 0x10 [constexpr]
```

simulation checking flag

Definition at line 25 of file [SimulationClass.h](#).

Referenced by [Simulation::advanceToTime\(\)](#), [Simulation::checkFlag\(\)](#), [Simulation::checkNoFlag\(\)](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::outAllFieldData\(\)](#), and [Simulation::start\(\)](#).

## 6.23 SimulationClass.h

[Go to the documentation of this file.](#)

```
00001 //////////////////////////////////////////////////////////////////
00002 /// @file SimulationClass.h
00003 /// @brief Class for the Simulation object calling all functionality:
00004 /// from wave settings over lattice construction, time evolution and outputs
00005 /// initialization of the CVode object
00006 //////////////////////////////////////////////////////////////////
00007
00008 #pragma once
00009
00010 /* access to the fixed point SUNNonlinearSolver */
00011 #include "sunnonlinsol/sunnonlinsol_fixedpoint.h"
00012
00013 // project subfile headers
00014 #include "ICSetters.h"
00015 #include "LatticePatch.h"
00016 #include "Outputters.h"
00017 #include "TimeEvolutionFunctions.h"
00018
00019 /**
00020 ** simulation checking flag */
00021 constexpr unsigned int LatticeDiscreteSetUp = 0x01;
00022 constexpr unsigned int LatticePhysicalSetUp = 0x02;
00023 constexpr unsigned int LatticePatchworkSetUp = 0x04; // not used anymore
00024 constexpr unsigned int CvodeObjectSetUp = 0x08;
00025 constexpr unsigned int SimulationStarted = 0x10;
00026 /**
00027
00028 /** @brief Simulation class to instantiate the whole walkthrough of a Simulation
00029 */
00030 class Simulation {
00031 private:
00032     /// Lattice object
00033     Lattice lattice;
00034     /// LatticePatch object
00035     LatticePatch latticePatch;
00036     /// current time of the simulation
00037     sunrealtype t;
00038     /// simulation status flags
00039     unsigned int statusFlags;
00040
00041 public:
00042     /// IC Setter object
00043     ICSetter icsettings;
00044     /// Output Manager object
00045     OutputManager outputManager;
00046     /// pointer to CVode memory object
00047     void *cvode_mem;
00048     /// nonlinear solver object
00049     SUNNonlinearSolver NLS;
00050     /// constructor function for the creation of the cartesian communicator
00051     Simulation(const int Nx, const int Ny, const int Nz, const int StencilOrder,
00052                 const bool periodicity);
00053     /// destructor function freeing CVode memory and Sundials context
00054     ~Simulation();
00055     /// reference to the cartesian communicator of the lattice (for debugging)
00056     MPI_Comm *get_cart_comm() { return &lattice.comm; }
00057     /// function to set discrete dimensions of the lattice
00058     void setDiscreteDimensionsOfLattice(const sunindextype _tot_nx,
00059                                         const sunindextype _tot_ny, const sunindextype _tot_nz);
00060     /// function to set physical dimensions of the lattice
00061     void setPhysicalDimensionsOfLattice(const sunrealtype lx,
00062                                         const sunrealtype ly, const sunrealtype lz);
```

```

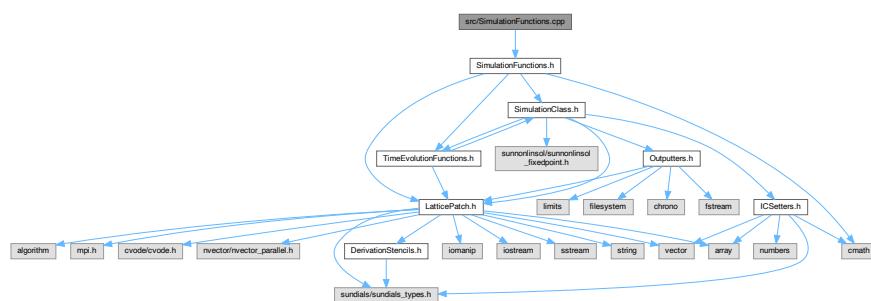
00063  /// function to initialize the Patchwork
00064  void initializePatchwork(const int nx, const int ny, const int nz);
00065  /// function to initialize the CVODE object with all requirements
00066  void initializeCVODEobject(const sunrealtype reltol,
00067           const sunrealtype abstol);
00068  /// function to start the simulation for time iteration
00069  void start();
00070  /// functions to set the initial field configuration onto the lattice
00071  void setInitialConditions();
00072  /// functions to add initial periodic field configurations
00073  void addInitialConditions(const sunindextype xm, const sunindextype ym,
00074           const sunindextype zm = 0);
00075  /// function to add a periodic IC layer in one dimension
00076  void addPeriodicICLayerInX();
00077  /// function to add periodic IC layers in two dimensions
00078  void addPeriodicICLayerInXY();
00079  /// function to advance solution in time with CVODE
00080  void advanceToTime(const sunrealtype &tEnd);
00081  /// function to write field data to disk
00082  void outAllFieldData(const int & state);
00083  /// function to check if flag has been set
00084  void checkFlag(unsigned int flag) const;
00085  /// function to check if flag has not been set
00086  // message and cause an abort on all ranks
00087  void checkNoFlag(unsigned int flag) const;
00088 };
00089

```

## 6.24 src/SimulationFunctions.cpp File Reference

Implementation of the complete simulation functions for 1D, 2D, and 3D, as called in the main function.

```
#include "SimulationFunctions.h"
Include dependency graph for SimulationFunctions.cpp:
```



## Functions

- void **timer** (double &t1, double &t2)
 

*MPI timer function.*
- void **Sim1D** (const std::array< sunrealtype, 2 > CVodeTol, const int StencilOrder, const sunrealtype phys\_← dim, const sunindextype disc\_dim, const bool periodic, int \*interactions, const sunrealtype endTime, const int numberOfSteps, const std::string outputDirectory, const int outputStep, const char outputStyle, const std← ::vector< **planewave** > &planes, const std::vector< **gaussian1D** > &gaussians)
 

*complete 1D Simulation function*
- void **Sim2D** (const std::array< sunrealtype, 2 > CVodeTol, int const StencilOrder, const std::array< sunrealtype, 2 > phys\_dims, const std::array< sunindextype, 2 > disc\_dims, const std::array< int, 2 > patches, const bool periodic, int \*interactions, const sunrealtype endTime, const int numberOfSteps, const std::string outputDirectory, const int outputStep, const char outputStyle, const std::vector< **planewave** > &planes, const std::vector< **gaussian2D** > &gaussians)

- void **Sim3D** (const std::array< sunrealtype, 2 > CVodeTol, const int StencilOrder, const std::array< sunrealtype, 3 > phys\_dims, const std::array< sunindextype, 3 > disc\_dims, const std::array< int, 3 > patches, const bool periodic, int \*interactions, const sunrealtype endTime, const int numberofSteps, const std::string outputDirectory, const int outputStep, const char outputStyle, const std::vector< planewave > &planes, const std::vector< gaussian3D > &gaussians)
- complete 3D Simulation function*

## 6.24.1 Detailed Description

Implementation of the complete simulation functions for 1D, 2D, and 3D, as called in the main function.

Definition in file [SimulationFunctions.cpp](#).

## 6.24.2 Function Documentation

### 6.24.2.1 Sim1D()

```
void Sim1D (
    const std::array< sunrealtype, 2 > CVodeTol,
    const int StencilOrder,
    const sunrealtype phys_dim,
    const sunindextype disc_dim,
    const bool periodic,
    int * interactions,
    const sunrealtype endTime,
    const int numberofSteps,
    const std::string outputDirectory,
    const int outputStep,
    const char outputStyle,
    const std::vector< planewave > & planes,
    const std::vector< gaussian1D > & gaussians )
```

complete 1D [Simulation function](#)

Conduct the complete 1D simulation process

Definition at line 21 of file [SimulationFunctions.cpp](#).

```
00028
00029
00030 // MPI data
00031 double ts = MPI_Wtime();
00032 int myPrc = 0, nPrc = 0;
00033 MPI_Comm_size(MPI_COMM_WORLD, &nPrc);
00034 MPI_Comm_rank(MPI_COMM_WORLD, &myPrc);
00035
00036 // Check feasibility of the patchwork decomposition
00037 if (myPrc == 0) {
00038     if (disc_dim % nPrc != 0) {
00039         errorKill("The number of lattice points must be "
00040                 "divisible by the number of processes.");
00041     }
00042 }
00043
00044 // Initialize the simulation, set up the cartesian communicator
00045 std::array<int, 3> patches = {nPrc, 1, 1};
00046 Simulation sim(patches[0], patches[1], patches[2], StencilOrder, periodic);
```

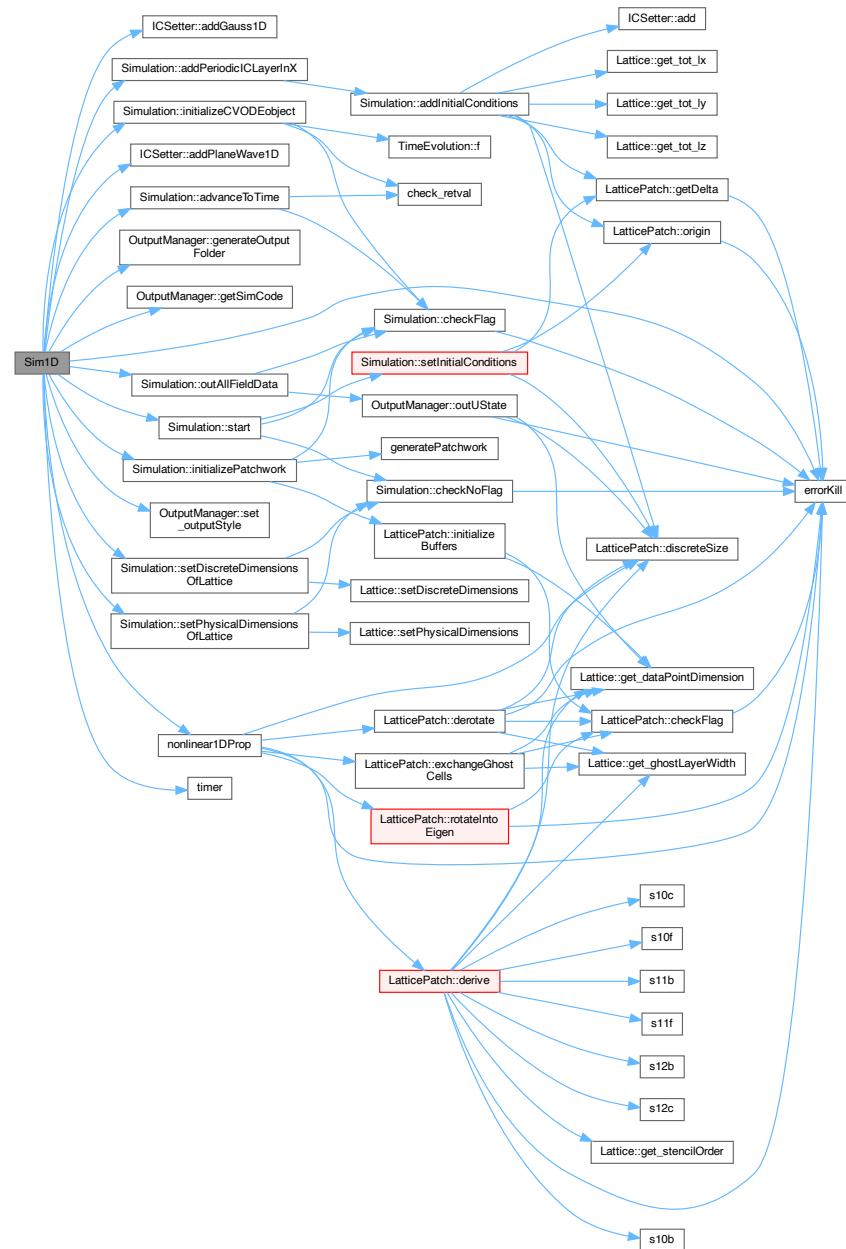
```

00047
00048 // Configure the patchwork
00049 sim.setPhysicalDimensionsOfLattice(phys_dim,1,1);
00050 sim.setDiscreteDimensionsOfLattice(disc_dim,1,1);
00051 sim.initializePatchwork(patches[0], patches[1], patches[2]);
00052
00053 // Add em-waves
00054 for (const auto &gauss : gaussians)
00055     sim.icsettings.addGauss1D(gauss.k, gauss.p, gauss.x0, gauss.phig,
00056                                 gauss.phi);
00057 for (const auto &plane : planes)
00058     sim.icsettings.addPlaneWave1D(plane.k, plane.p, plane.phi);
00059
00060 // Check that the patchwork is ready and set the initial conditions
00061 sim.start();
00062 sim.addPeriodicICLayerInX();
00063
00064 // Initialize CVode with abs and rel tolerances
00065 sim.initializeCVODEobject(CVodeTol[0], CVodeTol[1]);
00066
00067 // Configure the time evolution function
00068 TimeEvolution::c = interactions;
00069 TimeEvolution::TimeEvolver = nonlinear1DProp;
00070
00071 // Configure the output
00072 sim.outputManager.generateOutputFolder(outputDirectory);
00073 if (!myPrc) {
00074     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00075             << std::endl;
00076 }
00077 sim.outputManager.set_outputStyle(outputStyle);
00078
00079 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00080 //sim.outAllFieldData(0); // output of initial state
00081 // Conduct the propagation in space and time
00082 for (int step = 1; step <= numberOfSteps; step++) {
00083     sim.advanceToTime(endTime / numberOfSteps * step);
00084     if (step % outputStep == 0) {
00085         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00086         sim.outAllFieldData(step);
00087     }
00088     double tn = MPI_Wtime();
00089     if (!myPrc) {
00090         std::cout << "\rStep " << step << "\t\t" << std::flush;
00091         timer(ts, tn);
00092     }
00093 }
00094
00095 return;
00096 }
```

References [ICSetter::addGauss1D\(\)](#), [Simulation::addPeriodicICLayerInX\(\)](#), [ICSetter::addPlaneWave1D\(\)](#), [Simulation::advanceToTime\(\)](#), [TimeEvolution::c](#), [errorKill\(\)](#), [OutputManager::generateOutputFolder\(\)](#), [OutputManager::getSimCode\(\)](#), [Simulation::icsettings](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::initializePatchwork\(\)](#), [nonlinear1DProp\(\)](#), [Simulation::outAllFieldData\(\)](#), [Simulation::outputManager](#), [OutputManager::set\\_outputStyle\(\)](#), [Simulation::setDiscreteDimensionsOfLattice\(\)](#), [Simulation::setPhysicalDimensionsOfLattice\(\)](#), [Simulation::start\(\)](#), [TimeEvolution::TimeEvolver](#), and [timer\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.24.2.2 Sim2D()

```
void Sim2D (
    const std::array< sunrealtype, 2 > CVodeTol,
    int const StencilOrder,
    const std::array< sunrealtype, 2 > phys_dims,
    const std::array< sunindextype, 2 > disc_dims,
    const std::array< int, 2 > patches,
    const bool periodic,
    int * interactions,
    const sunrealtype endTime,
    const int numberofSteps,
    const std::string outputDirectory,
    const int outputStep,
    const char outputStyle,
    const std::vector< planewave > & planes,
    const std::vector< gaussian2D > & gaussians )
```

complete 2D Simulation function

Conduct the complete 2D simulation process

Definition at line 99 of file [SimulationFunctions.cpp](#).

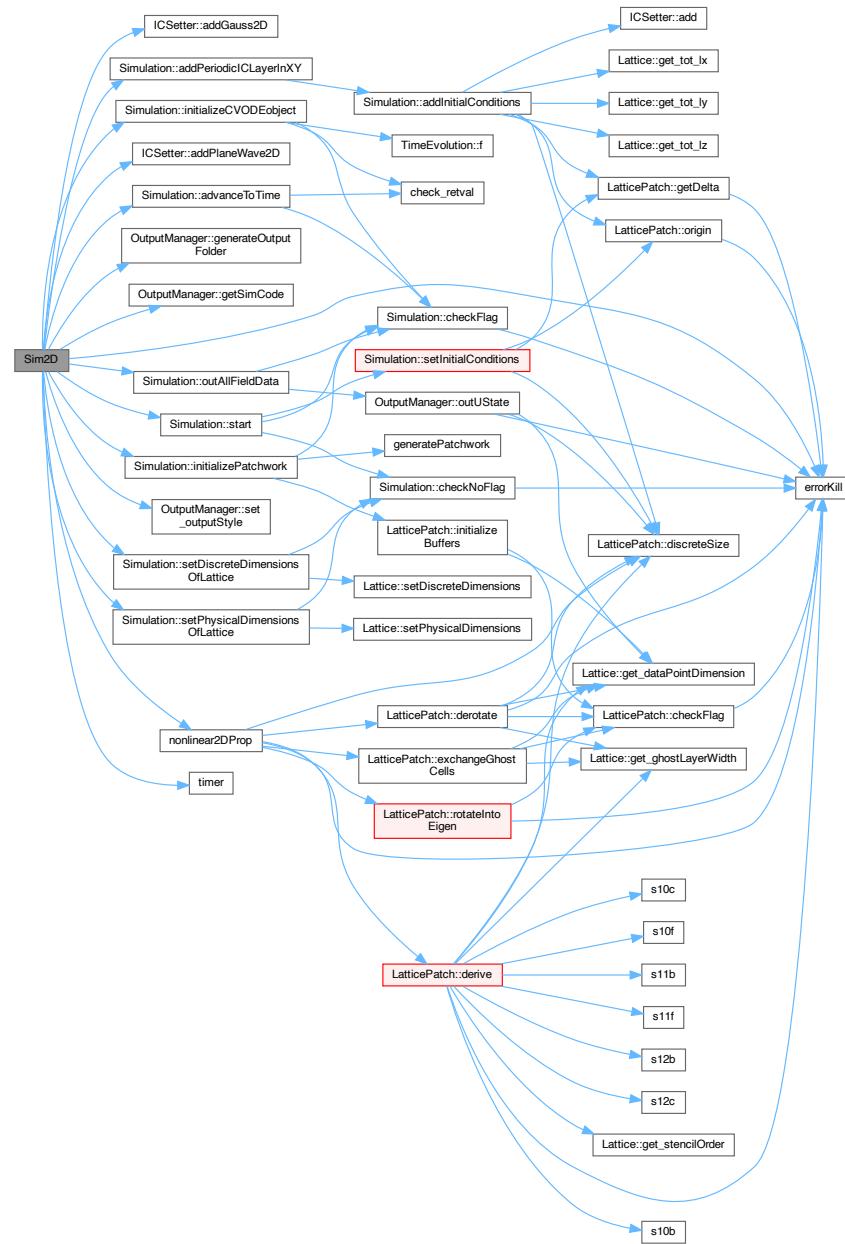
```
00107
00108
00109 // MPI data
00110 double ts = MPI_Wtime();
00111 int myPrc = 0, nPrc = 0; // Get process rank and number of processes
00112 MPI_Comm_rank(MPI_COMM_WORLD,
00113     &myPrc); // Return process rank, number \in [1,nPrc]
00114 MPI_Comm_size(MPI_COMM_WORLD,
00115     &nPrc); // Return number of processes (communicator size)
00116
00117 // Check feasibility of the patchwork decomposition
00118 if (myPrc == 0) {
00119     if (nPrc != patches[0] * patches[1]) {
00120         errorKill(
00121             "The number of MPI processes must match the number of patches.");
00122     }
00123 }
00124
00125 // Initialize the simulation, set up the cartesian communicator
00126 Simulation sim(patches[0], patches[1], 1, StencilOrder, periodic);
00127
00128 // Configure the patchwork
00129 sim.setPhysicalDimensionsOfLattice(phys_dims[0],
00130                                         phys_dims[1],
00131                                         1); // spacing of the lattice
00132 sim.setDiscreteDimensionsOfLattice(
00133     disc_dims[0], disc_dims[1], 1); // Spacing equivalence to points
00134 sim.initializePatchwork(patches[0], patches[1], 1);
00135
00136 // Add em-waves
00137 for (const auto &gauss : gaussians)
00138     sim.icsettings.addGauss2D(gauss.x0, gauss.axis, gauss.amp, gauss.phip,
00139                                gauss.w0, gauss.zr, gauss.ph0, gauss.phA);
00140 for (const auto &plane : planes)
00141     sim.icsettings.addPlaneWave2D(plane.k, plane.p, plane.phi);
00142
00143 // Check that the patchwork is ready and set the initial conditions
00144 sim.start(); // Check if the lattice is set up, set initial field
00145             // configuration
00146 sim.addPeriodicICLayerInXY(); // insure periodicity in propagation directions
00147
00148 // Initialize CVode with rel and abs tolerances
00149 sim.initializeCVODEobject(CVodeTol[0], CVodeTol[1]);
00150
00151 // Configure the time evolution function
00152 TimeEvolution::c = interactions;
00153 TimeEvolution::TimeEvolver = nonlinear2DProp;
00154
```

```
00155 // Configure the output
00156 sim.outputManager.generateOutputFolder(outputDirectory);
00157 if (!myPrc) {
00158     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00159     << std::endl;
00160 }
00161 sim.outputManager.set_outputStyle(outputStyle);
00162
00163 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00164 //sim.outAllFieldData(0); // output of initial state
00165 // Conduct the propagation in space and time
00166 for (int step = 1; step <= numberofSteps; step++) {
00167     sim.advanceToTime(endTime / numberofSteps * step);
00168     if (step % outputStep == 0) {
00169         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00170         sim.outAllFieldData(step);
00171     }
00172     double tn = MPI_Wtime();
00173     if (!myPrc) {
00174         std::cout << "\rStep " << step << "\t\t" << std::flush;
00175         timer(ts, tn);
00176     }
00177 }
00178
00179 return;
00180 }
```

References [ICSetter::addGauss2D\(\)](#), [Simulation::addPeriodicICLayerInXY\(\)](#), [ICSetter::addPlaneWave2D\(\)](#), [Simulation::advanceToTime\(\)](#), [TimeEvolution::c\\_errorKill\(\)](#), [OutputManager::generateOutputFolder\(\)](#), [OutputManager::getSimCode\(\)](#), [Simulation::icsettings](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::initializePatchwork\(\)](#), [nonlinear2DProp\(\)](#), [Simulation::outAllFieldData\(\)](#), [Simulation::outputManager](#), [OutputManager::set\\_outputStyle\(\)](#), [Simulation::setDiscreteDimensionsOfLattice\(\)](#), [Simulation::setPhysicalDimensionsOfLattice\(\)](#), [Simulation::start\(\)](#), [TimeEvolution::TimeEvolver](#), and [timer\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.24.2.3 Sim3D()

```
void Sim3D (
    const std::array< sunrealtype, 2 > CVodeTol,
    const int StencilOrder,
    const std::array< sunrealtype, 3 > phys_dims,
    const std::array< sunindextype, 3 > disc_dims,
    const std::array< int, 3 > patches,
    const bool periodic,
    int * interactions,
    const sunrealtype endTime,
    const int numberofSteps,
    const std::string outputDirectory,
    const int outputStep,
    const char outputStyle,
    const std::vector< planewave > & planes,
    const std::vector< gaussian3D > & gaussians )
```

complete 3D Simulation function

Conduct the complete 3D simulation process

Definition at line 183 of file [SimulationFunctions.cpp](#).

```
00191
00192
00193 // MPI data
00194 double ts = MPI_Wtime();
00195 int myPrc = 0, nPrc = 0; // Get process rank and numer of process
00196 MPI_Comm_rank(MPI_COMM_WORLD,
00197     &myPrc); // rank of the process inside the world communicator
00198 MPI_Comm_size(MPI_COMM_WORLD,
00199     &nPrc); // Size of the communicator is the number of processes
00200
00201 // Check feasibility of the patchwork decomposition
00202 if (myPrc == 0) {
00203     if (nPrc != patches[0] * patches[1] * patches[2]) {
00204         errorKill(
00205             "The number of MPI processes must match the number of patches.");
00206     }
00207     /*
00208     if ( ( disc_dims[0] / patches[0] != disc_dims[1] / patches[1] ) ||
00209         ( disc_dims[0] / patches[0] != disc_dims[2] / patches[2] ) ) {
00210         std::clog
00211             << "\nWarning: Patches should be cubic in terms of the lattice "
00212                 "points for the computational efficiency of larger simulations.\n";
00213     }
00214 */
00215 }
00216
00217 // Initialize the simulation, set up the cartesian communicator
00218 Simulation sim(patches[0], patches[1], patches[2],
00219                  StencilOrder, periodic); // simulation object with slicing
00220
00221 // Create the SUNContext object associated with the thread of execution
00222 sim.setPhysicalDimensionsOfLattice(phys_dims[0], phys_dims[1],
00223                                     phys_dims[2]); // spacing of the box
00224 sim.setDiscreteDimensionsOfLattice(
00225     disc_dims[0], disc_dims[1],
00226     disc_dims[2]); // Spacing equivalence to points
00227 sim.initializePatchwork(patches[0], patches[1], patches[2]);
00228
00229 // Add em-waves
00230 for (const auto &plane : planes)
00231     sim.icsettings.addPlaneWave3D(plane.k, plane.p, plane.phi);
00232 for (const auto &gauss : gaussians)
00233     sim.icsettings.addGauss3D(gauss.x0, gauss.axis, gauss.amp, gauss.phip,
00234                               gauss.w0, gauss.zr, gauss.ph0, gauss.phA);
00235
00236 // Check that the patchwork is ready and set the initial conditions
00237 sim.start();
00238
```

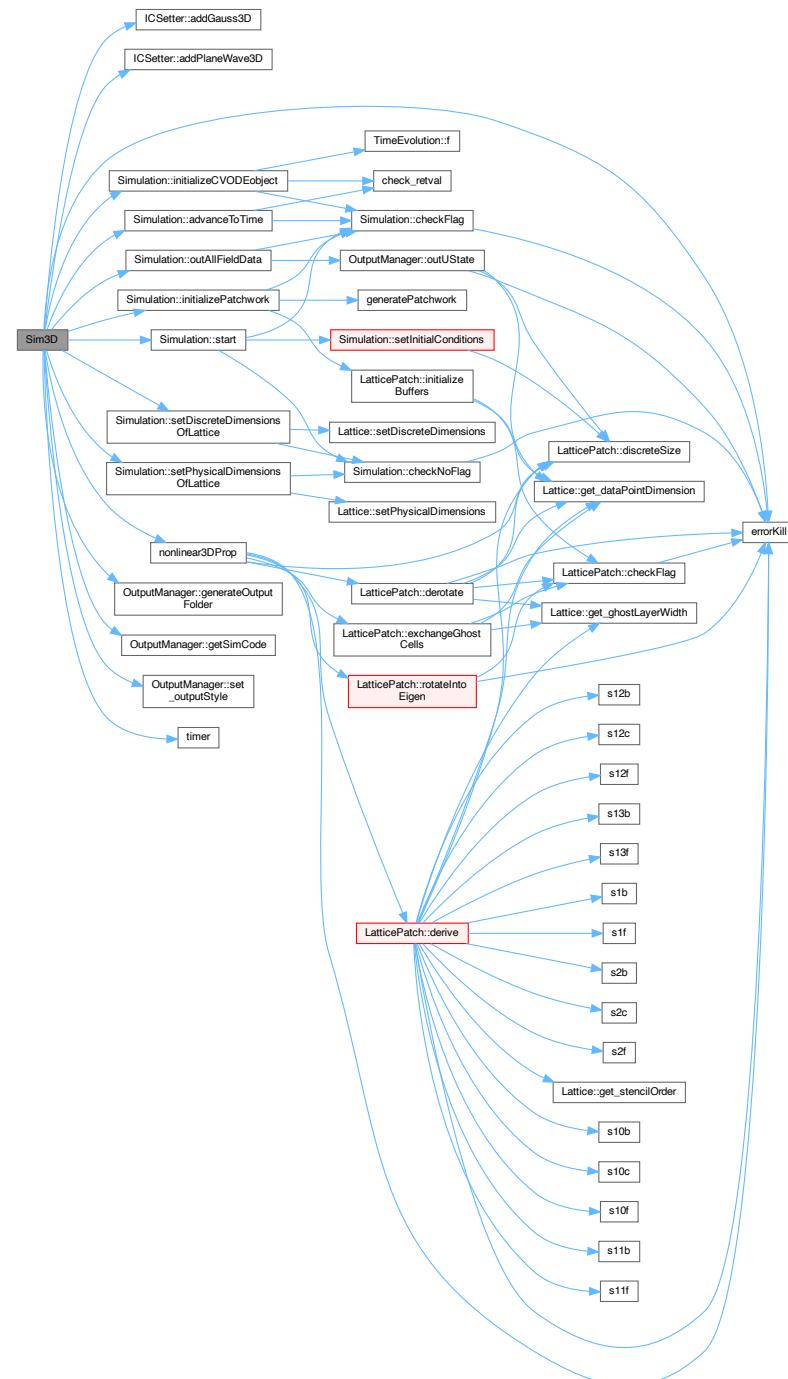
```

00239 // Initialize CVode with abs and rel tolerances
00240 sim.initializeCVODEobject(CVodeTol[0], CVodeTol[1]);
00241
00242 // Configure the time evolution function
00243 TimeEvolution::c = interactions;
00244 TimeEvolution::TimeEvolver = nonlinear3DProp;
00245
00246 // Configure the output
00247 sim.outputManager.generateOutputFolder(outputDirectory);
00248 if (!myPrc) {
00249     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00250         << std::endl;
00251 }
00252 sim.outputManager.set_outputStyle(outputStyle);
00253
00254 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00255 //sim.outAllFieldData(0); // output of initial state
00256 // Conduct the propagation in space and time
00257 for (int step = 1; step <= number_of_Steps; step++) {
00258     sim.advanceToTime(endTime / number_of_Steps * step);
00259     if (step % outputStep == 0) {
00260         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00261         sim.outAllFieldData(step);
00262     }
00263     double tn = MPI_Wtime();
00264     if (!myPrc) {
00265         std::cout << "\rStep " << step << "\t\t" << std::flush;
00266         timer(ts, tn);
00267     }
00268 }
00269 return;
00270 }
```

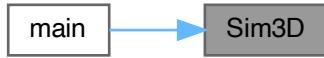
References [ICSetter::addGauss3D\(\)](#), [ICSetter::addPlaneWave3D\(\)](#), [Simulation::advanceToTime\(\)](#), [TimeEvolution::c](#), [errorKill\(\)](#), [OutputManager::generateOutputFolder\(\)](#), [OutputManager::getSimCode\(\)](#), [Simulation::icsettings](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::initializePatchwork\(\)](#), [nonlinear3DProp\(\)](#), [Simulation::outAllFieldData\(\)](#), [Simulation::outputManager](#), [OutputManager::set\\_outputStyle\(\)](#), [Simulation::setDiscreteDimensionsOfLattice\(\)](#), [Simulation::setPhysicalDimensionsOfLattice\(\)](#), [Simulation::start\(\)](#), [TimeEvolution::TimeEvolver](#), and [timer\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.24.2.4 timer()

```
void timer (
    double & t1,
    double & t2 ) [inline]
```

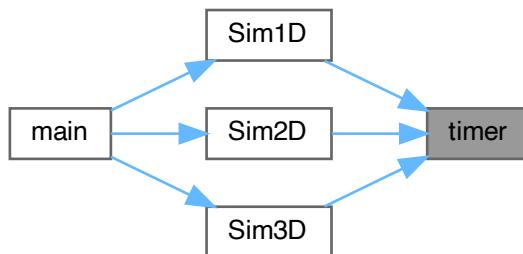
MPI timer function.

Calculate and print the total simulation time

Definition at line 10 of file [SimulationFunctions.cpp](#).  
00010  
00011   printf("Elapsed time: %fs\n", (t2 - t1));  
00012 }

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



## 6.25 SimulationFunctions.cpp

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file SimulationFunctions.cpp
00003 /// @brief Implementation of the complete simulation functions for
00004 /// 1D, 2D, and 3D, as called in the main function
00005 ///////////////////////////////////////////////////////////////////
00006
00007 #include "SimulationFunctions.h"
00008
00009 /** Calculate and print the total simulation time */
00010 inline void timer(double &t1, double &t2) {
00011     printf("Elapsed time: %fs\n", (t2 - t1));
00012 }
00013
00014 // Instantiate and preliminarily initialize the time evolver
00015 // non-const statics to be defined in actual simulation process
00016 int *TimeEvolution::c = nullptr;
00017 void (*TimeEvolution::TimeEvolver)(LatticePatch *, N_Vector, N_Vector,
00018                                     int *) = nonlinear1DProp;
00019
00020 /** Conduct the complete 1D simulation process */
00021 void Sim1D(const std::array<sunrealtype,2> CVodeTol, const int StencilOrder,
00022             const unrealtype phys_dim, const sunindextype disc_dim,
00023             const bool periodic, int *interactions,
00024             const unrealtype endTime, const int numberofSteps,
00025             const std::string outputDirectory, const int outputStep,
00026             const char outputStyle,
00027             const std::vector<planewave> &planes,
00028             const std::vector<gaussian1D> &gaussians) {
00029
00030     // MPI data
00031     double ts = MPI_Wtime();
00032     int myPrc = 0, nPrc = 0;
00033     MPI_Comm_size(MPI_COMM_WORLD, &nPrc);
00034     MPI_Comm_rank(MPI_COMM_WORLD, &myPrc);
00035
00036     // Check feasibility of the patchwork decomposition
00037     if (myPrc == 0) {
00038         if (disc_dim % nPrc != 0) {
00039             errorKill("The number of lattice points must be "
00040                       "divisible by the number of processes.");
00041         }
00042     }
00043
00044     // Initialize the simulation, set up the cartesian communicator
00045     std::array<int, 3> patches = {nPrc, 1, 1};
00046     Simulation sim(patches[0], patches[1], patches[2], StencilOrder, periodic);
00047
00048     // Configure the patchwork
00049     sim.setPhysicalDimensionsOfLattice(phys_dim,1,1);
00050     sim.setDiscreteDimensionsOfLattice(disc_dim,1,1);
00051     sim.initializePatchwork(patches[0], patches[1], patches[2]);
00052
00053     // Add em-waves
00054     for (const auto &gauss : gaussians)
00055         sim.icsettings.addGauss1D(gauss.k, gauss.p, gauss.x0, gauss.phig,
00056                                   gauss.phi);
00057     for (const auto &plane : planes)
00058         sim.icsettings.addPlaneWave1D(plane.k, plane.p, plane.phi);
00059
00060     // Check that the patchwork is ready and set the initial conditions
00061     sim.start();
00062     sim.addPeriodicICLayerInX();
00063
00064     // Initialize CVode with abs and rel tolerances
00065     sim.initializeCVODEObject(CVodeTol[0], CVodeTol[1]);
00066
00067     // Configure the time evolution function
00068     TimeEvolution::c = interactions;
00069     TimeEvolution::TimeEvolver = nonlinear1DProp;
00070
00071     // Configure the output
00072     sim.outputManager.generateOutputFolder(outputDirectory);
00073     if (!myPrc) {
00074         std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00075             << std::endl;
00076     }
00077     sim.outputManager.set_outputStyle(outputStyle);
00078
00079     //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00080     //sim.outAllFieldData(0); // output of initial state
00081     // Conduct the propagation in space and time
00082     for (int step = 1; step <= numberofSteps; step++) {

```

```

00083     sim.advanceToTime(endTime / number_of_steps * step);
00084     if (step % output_step == 0) {
00085         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00086         sim.outAllFieldData(step);
00087     }
00088     double tn = MPI_Wtime();
00089     if (!myPrc) {
00090         std::cout << "\rStep " << step << "\t\t" << std::flush;
00091         timer(ts, tn);
00092     }
00093 }
00094
00095 return;
00096 }
00097
00098 /** Conduct the complete 2D simulation process */
00099 void Sim2D(const std::array<sunrealtype,2> CVodeTol, int const StencilOrder,
00100     const std::array<sunrealtype,2> phys_dims,
00101     const std::array<sunindextype,2> disc_dims,
00102     const std::array<int,2> patches, const bool periodic, int *interactions,
00103     const unrealtype endTime, const int number_of_steps,
00104     const std::string output_directory, const int output_step,
00105     const char output_style,
00106     const std::vector<planewave> &planes,
00107     const std::vector<gaussian2D> &gaussians) {
00108
00109 // MPI data
00110 double ts = MPI_Wtime();
00111 int myPrc = 0, nPrc = 0; // Get process rank and number of processes
00112 MPI_Comm_rank(MPI_COMM_WORLD,
00113     &myPrc); // Return process rank, number \in [1,nPrc]
00114 MPI_Comm_size(MPI_COMM_WORLD,
00115     &nPrc); // Return number of processes (communicator size)
00116
00117 // Check feasibility of the patchwork decomposition
00118 if (myPrc == 0) {
00119     if (nPrc != patches[0] * patches[1]) {
00120         errorKill(
00121             "The number of MPI processes must match the number of patches.");
00122     }
00123 }
00124
00125 // Initialize the simulation, set up the cartesian communicator
00126 Simulation sim(patches[0], patches[1], 1, StencilOrder, periodic);
00127
00128 // Configure the patchwork
00129 sim.setPhysicalDimensionsOfLattice(phys_dims[0],
00130     phys_dims[1],
00131     1); // spacing of the lattice
00132 sim.setDiscreteDimensionsOfLattice(
00133     disc_dims[0], disc_dims[1], 1); // Spacing equivalence to points
00134 sim.initializePatchwork(patches[0], patches[1], 1);
00135
00136 // Add em-waves
00137 for (const auto &gauss : gaussians)
00138     sim.icsettings.addGauss2D(gauss.x0, gauss.axis, gauss.amp, gauss.phip,
00139         gauss.w0, gauss.zr, gauss.ph0, gauss.phA);
00140 for (const auto &plane : planes)
00141     sim.icsettings.addPlaneWave2D(plane.k, plane.p, plane.phi);
00142
00143 // Check that the patchwork is ready and set the initial conditions
00144 sim.start(); // Check if the lattice is set up, set initial field
00145             // configuration
00146 sim.addPeriodicICLayerInXY(); // insure periodicity in propagation directions
00147
00148 // Initialize CVode with rel and abs tolerances
00149 sim.initializeCVODEObject(CVodeTol[0], CVodeTol[1]);
00150
00151 // Configure the time evolution function
00152 TimeEvolution::c = interactions;
00153 TimeEvolution::TimeEvolver = nonlinear2DProp;
00154
00155 // Configure the output
00156 sim.outputManager.generateOutputFolder(output_directory);
00157 if (!myPrc) {
00158     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00159         << std::endl;
00160 }
00161 sim.outputManager.setOutputStyle(output_style);
00162
00163 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00164 //sim.outAllFieldData(0); // output of initial state
00165 // Conduct the propagation in space and time
00166 for (int step = 1; step <= number_of_steps; step++) {
00167     sim.advanceToTime(endTime / number_of_steps * step);
00168     if (step % output_step == 0) {
00169         MPI_Barrier(MPI_COMM_WORLD); // insure correct output

```

```

00170     sim.outAllFieldData(step);
00171 }
00172 double tn = MPI_Wtime();
00173 if (!myPrc) {
00174     std::cout << "\rStep " << step << "\t\t" << std::flush;
00175     timer(ts, tn);
00176 }
00177 }
00178
00179 return;
00180 }
00181
00182 /** Conduct the complete 3D simulation process */
00183 void Sim3D(const std::array<sunrealtype,2> CVodeTol, const int StencilOrder,
00184             const std::array<sunrealtype,3> phys_dims,
00185             const std::array<sunindextype,3> disc_dims,
00186             const std::array<int,3> patches,
00187             const bool periodic, int *interactions, const unrealtype endTime,
00188             const int numberOfSteps, const std::string outputDirectory,
00189             const int outputStep, const char outputStyle,
00190             const std::vector<planewave> &planes,
00191             const std::vector<gaussian3D> &gaussians) {
00192
00193 // MPI data
00194 double ts = MPI_Wtime();
00195 int myPrc = 0, nPrc = 0; // Get process rank and numer of process
00196 MPI_Comm_rank(MPI_COMM_WORLD,
00197                 &myPrc); // rank of the process inside the world communicator
00198 MPI_Comm_size(MPI_COMM_WORLD,
00199                 &nPrc); // Size of the communicator is the number of processes
00200
00201 // Check feasibility of the patchwork decomposition
00202 if (myPrc == 0) {
00203     if (nPrc != patches[0] * patches[1] * patches[2]) {
00204         errorKill(
00205             "The number of MPI processes must match the number of patches.");
00206     }
00207     /*
00208     if ( ( disc_dims[0] / patches[0] != disc_dims[1] / patches[1] ) ||
00209         ( disc_dims[0] / patches[0] != disc_dims[2] / patches[2] ) ) {
00210         std::clog
00211             << "\nWarning: Patches should be cubic in terms of the lattice "
00212             "points for the computational efficiency of larger simulations.\n";
00213     }
00214     */
00215 }
00216
00217 // Initialize the simulation, set up the cartesian communicator
00218 Simulation sim(patches[0], patches[1], patches[2],
00219                  StencilOrder, periodic); // Simulation object with slicing
00220
00221 // Create the SUNContext object associated with the thread of execution
00222 sim.setPhysicalDimensionsOfLattice(phys_dims[0], phys_dims[1],
00223                                     phys_dims[2]); // spacing of the box
00224 sim.setDiscreteDimensionsOfLattice(
00225     disc_dims[0], disc_dims[1],
00226     disc_dims[2]); // Spacing equivalence to points
00227 sim.initializePatchwork(patches[0], patches[1], patches[2]);
00228
00229 // Add em-waves
00230 for (const auto &plane : planes)
00231     sim.icsettings.addPlaneWave3D(plane.k, plane.p, plane.phi);
00232 for (const auto &gauss : gaussians)
00233     sim.icsettings.addGauss3D(gauss.x0, gauss.axis, gauss.amp, gauss.phip,
00234                               gauss.w0, gauss.zr, gauss.ph0, gauss.phA);
00235
00236 // Check that the patchwork is ready and set the initial conditions
00237 sim.start();
00238
00239 // Initialize CVode with abs and rel tolerances
00240 sim.initializeCVODEObject(CVodeTol[0], CVodeTol[1]);
00241
00242 // Configure the time evolution function
00243 TimeEvolution::c = interactions;
00244 TimeEvolution::TimeEvolver = nonlinear3DProp;
00245
00246 // Configure the output
00247 sim.outputManager.generateOutputFolder(outputDirectory);
00248 if (!myPrc) {
00249     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00250             << std::endl;
00251 }
00252 sim.outputManager.setOutputStyle(outputStyle);
00253
00254 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00255 //sim.outAllFieldData(0); // output of initial state
00256 // Conduct the propagation in space and time

```

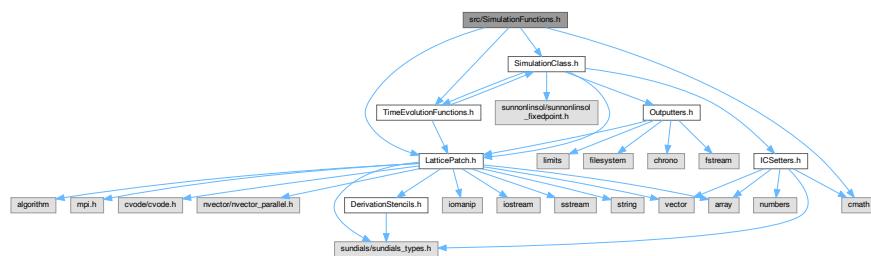
```

00257   for (int step = 1; step <= number_of_steps; step++) {
00258     sim.advance_to_time(end_time / number_of_steps * step);
00259     if (step % output_step == 0) {
00260       MPI_BARRIER(MPI_COMM_WORLD); // insure correct output
00261       sim.out_all_field_data(step);
00262     }
00263     double tn = MPI_Wtime();
00264     if (!my_prc) {
00265       std::cout << "\rStep " << step << "\t\t" << std::flush;
00266       timer(ts, tn);
00267     }
00268   }
00269   return;
00270 }
```

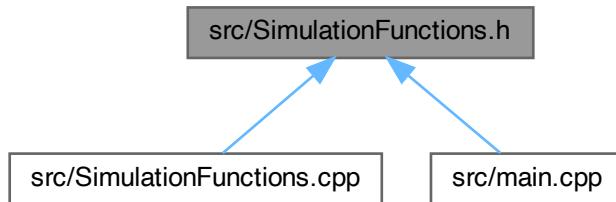
## 6.26 src/SimulationFunctions.h File Reference

Full simulation functions for 1D, 2D, and 3D used in [main.cpp](#).

```
#include <cmath>
#include "LatticePatch.h"
#include "SimulationClass.h"
#include "TimeEvolutionFunctions.h"
Include dependency graph for SimulationFunctions.h:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [planewave](#)  
*plane wave structure*

- struct `gaussian1D`  
*1D Gaussian wave structure*
- struct `gaussian2D`  
*2D Gaussian wave structure*
- struct `gaussian3D`  
*3D Gaussian wave structure*

## Functions

- void `Sim1D` (const std::array< sunrealtype, 2 >, const int, const sunrealtype, const sunindextype, const bool, int \*, const sunrealtype, const int, const std::string, const int, const char, const std::vector< `planewave` > &, const std::vector< `gaussian1D` > &)  
*complete 1D Simulation function*
- void `Sim2D` (const std::array< sunrealtype, 2 >, const int, const std::array< sunrealtype, 2 >, const std::array< sunindextype, 2 >, const std::array< int, 2 >, const bool, int \*, const sunrealtype, const int, const std::string, const int, const char, const std::vector< `planewave` > &, const std::vector< `gaussian2D` > &)  
*complete 2D Simulation function*
- void `Sim3D` (const std::array< sunrealtype, 2 >, const int, const std::array< sunrealtype, 3 >, const std::array< sunindextype, 3 >, const std::array< int, 3 >, const bool, int \*, const sunrealtype, const int, const std::string, const int, const char, const std::vector< `planewave` > &, const std::vector< `gaussian3D` > &)  
*complete 3D Simulation function*
- void `timer` (double &, double &)  
*MPI timer function.*

### 6.26.1 Detailed Description

Full simulation functions for 1D, 2D, and 3D used in `main.cpp`.

Definition in file `SimulationFunctions.h`.

### 6.26.2 Function Documentation

#### 6.26.2.1 Sim1D()

```
void Sim1D (
    const std::array< sunrealtype, 2 > CVodeTol,
    const int StencilOrder,
    const sunrealtype phys_dim,
    const sunindextype disc_dim,
    const bool periodic,
    int * interactions,
    const sunrealtype endTime,
    const int numberOfSteps,
    const std::string outputDirectory,
    const int outputStep,
    const char outputStyle,
```

```
    const std::vector< planewave > & planes,
    const std::vector< gaussian1D > & gaussians )
```

complete 1D Simulation function

Conduct the complete 1D simulation process

Definition at line 21 of file [SimulationFunctions.cpp](#).

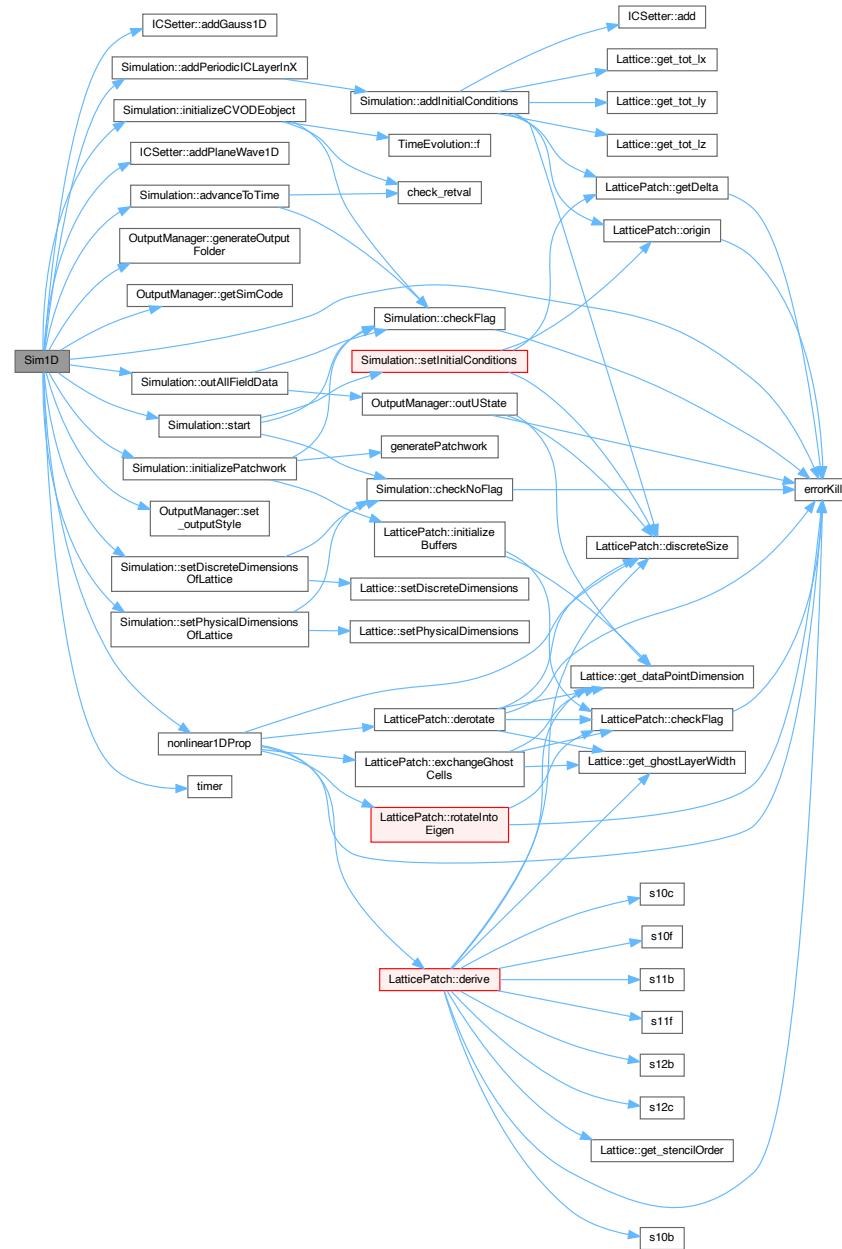
```
00028
00029
00030 // MPI data
00031 double ts = MPI_Wtime();
00032 int myPrc = 0, nPrc = 0;
00033 MPI_Comm_size(MPI_COMM_WORLD, &nPrc);
00034 MPI_Comm_rank(MPI_COMM_WORLD, &myPrc);
00035
00036 // Check feasibility of the patchwork decomposition
00037 if (myPrc == 0) {
00038     if (disc_dim % nPrc != 0) {
00039         errorKill("The number of lattice points must be "
00040                 "divisible by the number of processes.");
00041     }
00042 }
00043
00044 // Initialize the simulation, set up the cartesian communicator
00045 std::array<int, 3> patches = {nPrc, 1, 1};
00046 Simulation sim(patches[0], patches[1], patches[2], StencilOrder, periodic);
00047
00048 // Configure the patchwork
00049 sim.setPhysicalDimensionsOfLattice(phys_dim,1,1);
00050 sim.setDiscreteDimensionsOfLattice(disc_dim,1,1);
00051 sim.initializePatchwork(patches[0], patches[1], patches[2]);
00052
00053 // Add em-waves
00054 for (const auto &gauss : gaussians)
00055     sim.icsettings.addGauss1D(gauss.k, gauss.p, gauss.x0, gauss.phig,
00056                               gauss.phi);
00057 for (const auto &plane : planes)
00058     sim.icsettings.addPlaneWave1D(plane.k, plane.p, plane.phi);
00059
00060 // Check that the patchwork is ready and set the initial conditions
00061 sim.start();
00062 sim.addPeriodicICLayerInX();
00063
00064 // Initialize CVode with abs and rel tolerances
00065 sim.initializeCVODEobject(CVodeTol[0], CVodeTol[1]);
00066
00067 // Configure the time evolution function
00068 TimeEvolution::c = interactions;
00069 TimeEvolution::TimeEvolver = nonlinear1DProp;
00070
00071 // Configure the output
00072 sim.outputManager.generateOutputFolder(outputDirectory);
00073 if (!myPrc) {
00074     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00075             << std::endl;
00076 }
00077 sim.outputManager.setOutputStyle(outputStyle);
00078
00079 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00080 //sim.outAllFieldData(0); // output of initial state
00081 // Conduct the propagation in space and time
00082 for (int step = 1; step <= number_of_steps; step++) {
00083     sim.advanceToTime(end_time / number_of_steps * step);
00084     if (step % output_step == 0) {
00085         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00086         sim.outAllFieldData(step);
00087     }
00088     double tn = MPI_Wtime();
00089     if (!myPrc) {
00090         std::cout << "\rStep " << step << "\t\t" << std::flush;
00091         timer(ts, tn);
00092     }
00093 }
00094
00095 return;
00096 }
```

References [ICSetter::addGauss1D\(\)](#), [Simulation::addPeriodicICLayerInX\(\)](#), [ICSetter::addPlaneWave1D\(\)](#), [Simulation::advanceToTime\(\)](#), [TimeEvolution::c](#), [errorKill\(\)](#), [OutputManager::generateOutputFolder\(\)](#), [OutputManager::getSimCode\(\)](#), [Simulation::icsettings](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::initializePatchwork\(\)](#), [nonlinear1DProp\(\)](#),

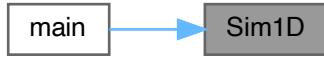
`Simulation::outAllFieldData()`, `Simulation::outputManager`, `OutputManager::set_outputStyle()`, `Simulation::setDiscreteDimensionsOfLattice()`, `Simulation::setPhysicalDimensionsOfLattice()`, `Simulation::start()`, `TimeEvolution::TimeEvolver`, and `timer()`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.26.2.2 Sim2D()

```

void Sim2D (
    const std::array< sunrealtype, 2 > CVodeTol,
    int const StencilOrder,
    const std::array< sunrealtype, 2 > phys_dims,
    const std::array< sunindextype, 2 > disc_dims,
    const std::array< int, 2 > patches,
    const bool periodic,
    int * interactions,
    const sunrealtype endTime,
    const int numberofSteps,
    const std::string outputDirectory,
    const int outputStep,
    const char outputStyle,
    const std::vector< planewave > & planes,
    const std::vector< gaussian2D > & gaussians )

```

complete 2D Simulation function

Conduct the complete 2D simulation process

Definition at line 99 of file [SimulationFunctions.cpp](#).

```

00107
00108
00109 // MPI data
00110 double ts = MPI_Wtime();
00111 int myPrc = 0, nPrc = 0; // Get process rank and number of processes
00112 MPI_Comm_rank(MPI_COMM_WORLD,
00113                 &myPrc); // Return process rank, number \in [1,nPrc]
00114 MPI_Comm_size(MPI_COMM_WORLD,
00115                 &nPrc); // Return number of processes (communicator size)
00116
00117 // Check feasibility of the patchwork decomposition
00118 if (myPrc == 0) {
00119     if (nPrc != patches[0] * patches[1]) {
00120         errorKill(
00121             "The number of MPI processes must match the number of patches.");
00122     }
00123 }
00124
00125 // Initialize the simulation, set up the cartesian communicator
00126 Simulation sim(patches[0], patches[1], 1, StencilOrder, periodic);
00127
00128 // Configure the patchwork
00129 sim.setPhysicalDimensionsOfLattice(phys_dims[0],
00130                                     phys_dims[1],
00131                                     1); // spacing of the lattice
00132 sim.setDiscreteDimensionsOfLattice(
00133     disc_dims[0], disc_dims[1], 1); // Spacing equivalence to points
00134 sim.initializePatchwork(patches[0], patches[1], 1);

```

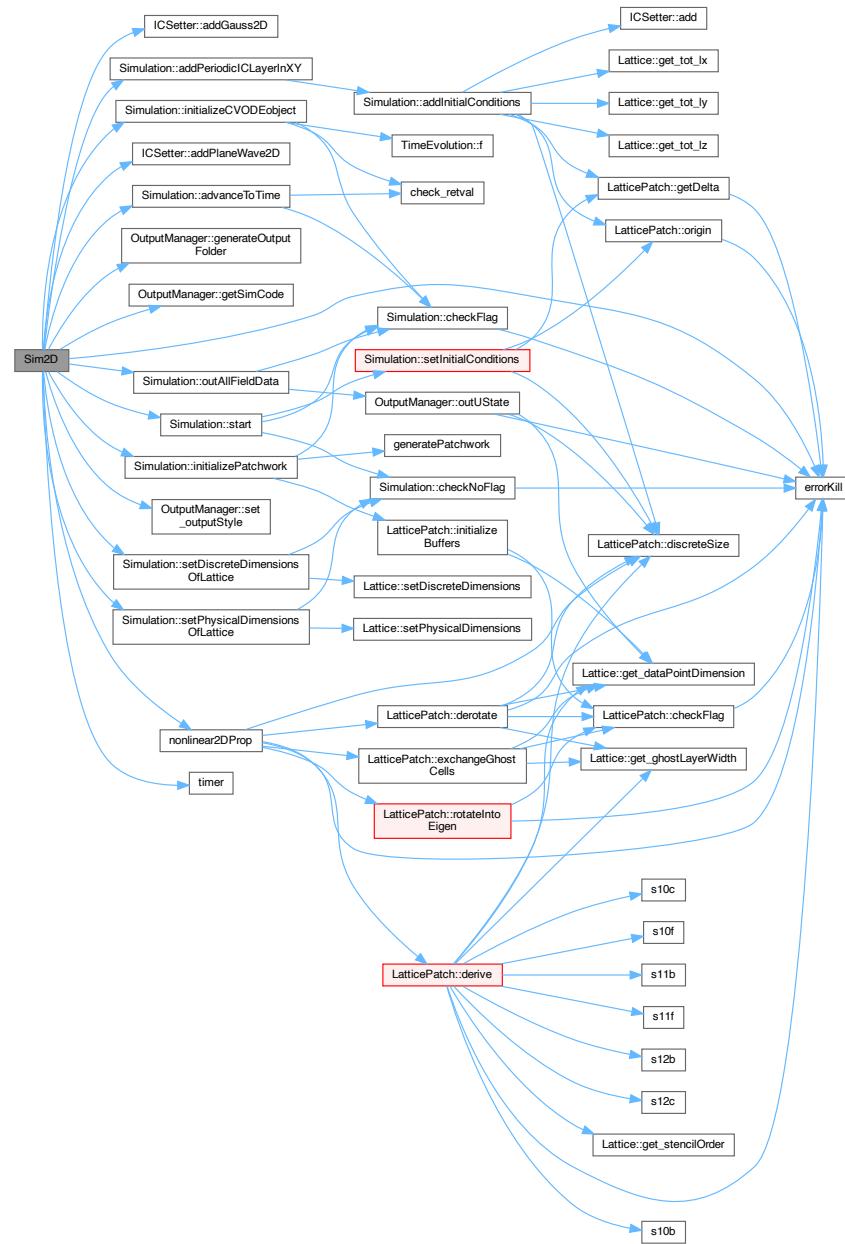
```

00135
00136 // Add em-waves
00137 for (const auto &gauss : gaussians)
00138     sim.icsettings.addGauss2D(gauss.x0, gauss.axis, gauss.amp, gauss.phip,
00139                             gauss.w0, gauss.zr, gauss.ph0, gauss.phA);
00140 for (const auto &plane : planes)
00141     sim.icsettings.addPlaneWave2D(plane.k, plane.p, plane.phi);
00142
00143 // Check that the patchwork is ready and set the initial conditions
00144 sim.start(); // Check if the lattice is set up, set initial field
00145             // configuration
00146 sim.addPeriodicICLayerInXY(); // insure periodicity in propagation directions
00147
00148 // Initialize CVode with rel and abs tolerances
00149 sim.initializeCVODEobject(CVodeTol[0], CVodeTol[1]);
00150
00151 // Configure the time evolution function
00152 TimeEvolution::c = interactions;
00153 TimeEvolution::TimeEvolver = nonlinear2DProp;
00154
00155 // Configure the output
00156 sim.outputManager.generateOutputFolder(outputDirectory);
00157 if (!myPrc) {
00158     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00159             << std::endl;
00160 }
00161 sim.outputManager.set_outputStyle(outputStyle);
00162
00163 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00164 //sim.outAllFieldData(0); // output of initial state
00165 // Conduct the propagation in space and time
00166 for (int step = 1; step <= numberOfSteps; step++) {
00167     sim.advanceToTime(endTime / numberOfSteps * step);
00168     if (step % outputStep == 0) {
00169         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00170         sim.outAllFieldData(step);
00171     }
00172     double tn = MPI_Wtime();
00173     if (!myPrc) {
00174         std::cout << "\rStep " << step << "\t\t" << std::flush;
00175         timer(ts, tn);
00176     }
00177 }
00178 return;
00179 }
```

References [ICSetter::addGauss2D\(\)](#), [Simulation::addPeriodicICLayerInXY\(\)](#), [ICSetter::addPlaneWave2D\(\)](#), [Simulation::advanceToTime\(\)](#), [TimeEvolution::c](#), [errorKill\(\)](#), [OutputManager::generateOutputFolder\(\)](#), [OutputManager::getSimCode\(\)](#), [Simulation::icsettings](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::initializePatchwork\(\)](#), [nonlinear2DProp\(\)](#), [Simulation::outAllFieldData\(\)](#), [Simulation::outputManager](#), [OutputManager::set\\_outputStyle\(\)](#), [Simulation::setDiscreteDimensionsOfLattice\(\)](#), [Simulation::setPhysicalDimensionsOfLattice\(\)](#), [Simulation::start\(\)](#), [TimeEvolution::TimeEvolver](#), and [timer\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.26.2.3 Sim3D()

```
void Sim3D (
    const std::array< sunrealtype, 2 > CVodeTol,
    const int StencilOrder,
    const std::array< sunrealtype, 3 > phys_dims,
    const std::array< sunindextype, 3 > disc_dims,
    const std::array< int, 3 > patches,
    const bool periodic,
    int * interactions,
    const sunrealtype endTime,
    const int numberofSteps,
    const std::string outputDirectory,
    const int outputStep,
    const char outputStyle,
    const std::vector< planewave > & planes,
    const std::vector< gaussian3D > & gaussians )
```

complete 3D Simulation function

Conduct the complete 3D simulation process

Definition at line 183 of file [SimulationFunctions.cpp](#).

```
00191
00192
00193 // MPI data
00194 double ts = MPI_Wtime();
00195 int myPrc = 0, nPrc = 0; // Get process rank and numer of process
00196 MPI_Comm_rank(MPI_COMM_WORLD,
00197     &myPrc); // rank of the process inside the world communicator
00198 MPI_Comm_size(MPI_COMM_WORLD,
00199     &nPrc); // Size of the communicator is the number of processes
00200
00201 // Check feasibility of the patchwork decomposition
00202 if (myPrc == 0) {
00203     if (nPrc != patches[0] * patches[1] * patches[2]) {
00204         errorKill(
00205             "The number of MPI processes must match the number of patches.");
00206     }
00207     /*
00208     if ( ( disc_dims[0] / patches[0] != disc_dims[1] / patches[1] ) ||
00209         ( disc_dims[0] / patches[0] != disc_dims[2] / patches[2] ) ) {
00210         std::clog
00211             << "\nWarning: Patches should be cubic in terms of the lattice "
00212                 "points for the computational efficiency of larger simulations.\n";
00213     }
00214 */
00215 }
00216
00217 // Initialize the simulation, set up the cartesian communicator
00218 Simulation sim(patches[0], patches[1], patches[2],
00219                  StencilOrder, periodic); // simulation object with slicing
00220
00221 // Create the SUNContext object associated with the thread of execution
00222 sim.setPhysicalDimensionsOfLattice(phys_dims[0], phys_dims[1],
00223                                     phys_dims[2]); // spacing of the box
00224 sim.setDiscreteDimensionsOfLattice(
00225     disc_dims[0], disc_dims[1],
00226     disc_dims[2]); // Spacing equivalence to points
00227 sim.initializePatchwork(patches[0], patches[1], patches[2]);
00228
00229 // Add em-waves
00230 for (const auto &plane : planes)
00231     sim.icsettings.addPlaneWave3D(plane.k, plane.p, plane.phi);
00232 for (const auto &gauss : gaussians)
00233     sim.icsettings.addGauss3D(gauss.x0, gauss.axis, gauss.amp, gauss.phip,
00234                               gauss.w0, gauss.zr, gauss.ph0, gauss.phA);
00235
00236 // Check that the patchwork is ready and set the initial conditions
00237 sim.start();
00238
```

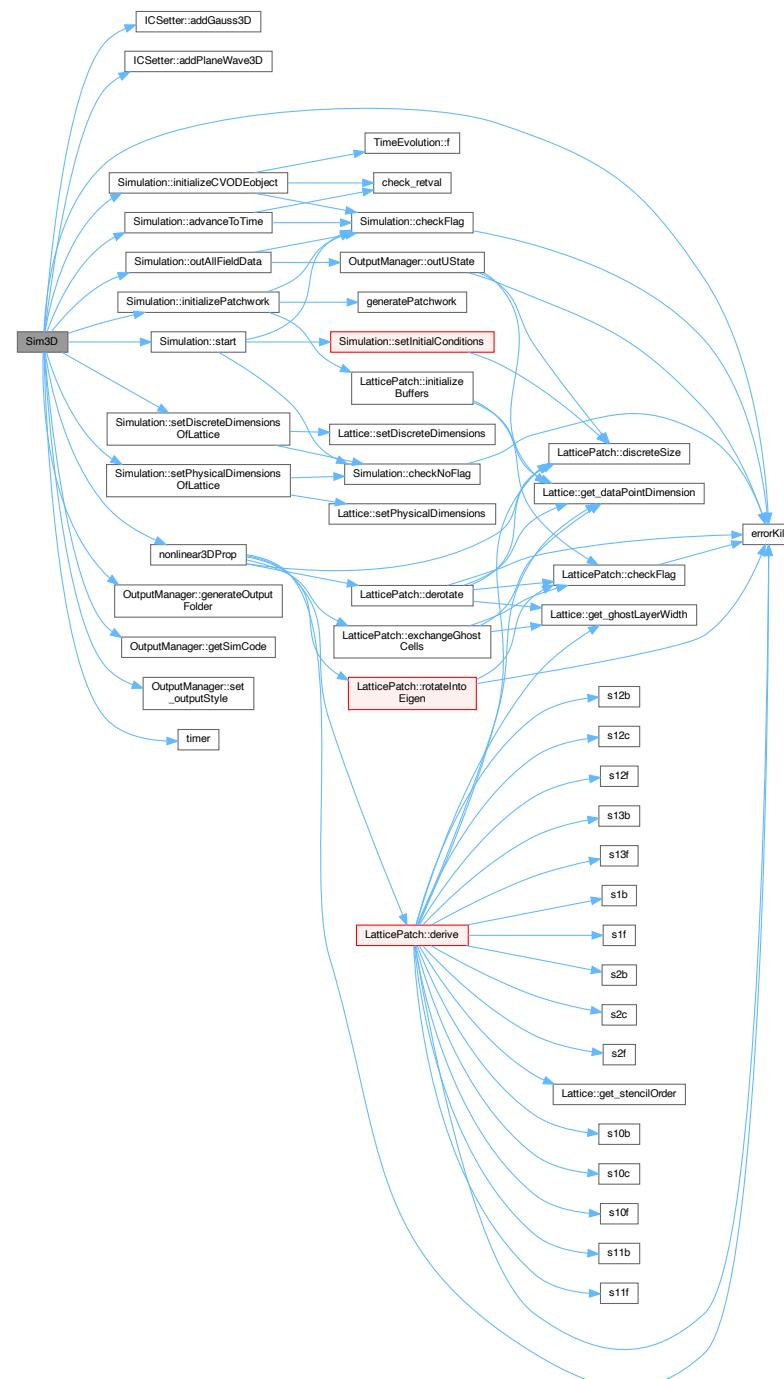
```

00239 // Initialize CVode with abs and rel tolerances
00240 sim.initializeCVODEobject(CVodeTol[0], CVodeTol[1]);
00241
00242 // Configure the time evolution function
00243 TimeEvolution::c = interactions;
00244 TimeEvolution::TimeEvolver = nonlinear3DProp;
00245
00246 // Configure the output
00247 sim.outputManager.generateOutputFolder(outputDirectory);
00248 if (!myPrc) {
00249     std::cout << "Simulation code: " << sim.outputManager.getSimCode()
00250         << std::endl;
00251 }
00252 sim.outputManager.set_outputStyle(outputStyle);
00253
00254 //MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00255 //sim.outAllFieldData(0); // output of initial state
00256 // Conduct the propagation in space and time
00257 for (int step = 1; step <= number_of_Steps; step++) {
00258     sim.advanceToTime(endTime / number_of_Steps * step);
00259     if (step % outputStep == 0) {
00260         MPI_Barrier(MPI_COMM_WORLD); // insure correct output
00261         sim.outAllFieldData(step);
00262     }
00263     double tn = MPI_Wtime();
00264     if (!myPrc) {
00265         std::cout << "\rStep " << step << "\t\t" << std::flush;
00266         timer(ts, tn);
00267     }
00268 }
00269 return;
00270 }
```

References [ICSetter::addGauss3D\(\)](#), [ICSetter::addPlaneWave3D\(\)](#), [Simulation::advanceToTime\(\)](#), [TimeEvolution::c](#), [errorKill\(\)](#), [OutputManager::generateOutputFolder\(\)](#), [OutputManager::getSimCode\(\)](#), [Simulation::icsettings](#), [Simulation::initializeCVODEobject\(\)](#), [Simulation::initializePatchwork\(\)](#), [nonlinear3DProp\(\)](#), [Simulation::outAllFieldData\(\)](#), [Simulation::outputManager](#), [OutputManager::set\\_outputStyle\(\)](#), [Simulation::setDiscreteDimensionsOfLattice\(\)](#), [Simulation::setPhysicalDimensionsOfLattice\(\)](#), [Simulation::start\(\)](#), [TimeEvolution::TimeEvolver](#), and [timer\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.26.2.4 timer()

```
void timer (
    double & t1,
    double & t2 ) [inline]
```

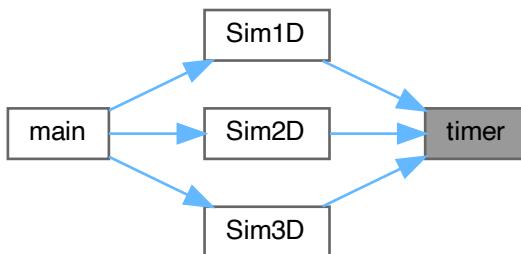
MPI timer function.

Calculate and print the total simulation time

Definition at line 10 of file [SimulationFunctions.cpp](#).  
00010  
00011   printf("Elapsed time: %fs\n", (t2 - t1));  
00012 }

Referenced by [Sim1D\(\)](#), [Sim2D\(\)](#), and [Sim3D\(\)](#).

Here is the caller graph for this function:



## 6.27 SimulationFunctions.h

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file SimulationFunctions.h
00003 /// @brief Full simulation functions for 1D, 2D, and 3D used in main.cpp
00004 ///////////////////////////////////////////////////////////////////
00005
00006 #pragma once
00007
00008 // math
00009 #include <cmath>
00010
00011 // project subfile headers
00012 #include "LatticePatch.h"
00013 #include "SimulationClass.h"
00014 #include "TimeEvolutionFunctions.h"
00015
00016 /***** EM-wave structures *****/
00017
00018 /// plane wave structure
00019 struct planewave {
00020     std::array<sunrealtyp, 3> k;    /**< wavevector (normalized to \f$ 1/\lambda \f$) */
00021     std::array<sunrealtyp, 3> p;    /**< amplitude & polarization vector */
00022     std::array<sunrealtyp, 3> phi;  /**< phase shift */
00023 };
00024
00025 /// 1D Gaussian wave structure
00026 struct gaussian1D {
00027     std::array<sunrealtyp, 3> k;    /**< wavevector (normalized to \f$ 1/\lambda \f$) */
00028     std::array<sunrealtyp, 3> p;    /**< amplitude & polarization vector */
00029     std::array<sunrealtyp, 3> x0;   /**< shift from origin */
00030     sunrealtyp phig;            /**< width */
00031     std::array<sunrealtyp, 3> phi; /**< phase shift */
00032 };
00033
00034 /// 2D Gaussian wave structure
00035 struct gaussian2D {
00036     std::array<sunrealtyp, 3> x0;   /**< center */
00037     std::array<sunrealtyp, 3> axis; /**< direction from where it comes */
00038     sunrealtyp amp;              /**< amplitude */
00039     sunrealtyp phip;             /**< polarization rotation */
00040     sunrealtyp w0;               /**< taille */
00041     sunrealtyp zr;               /**< Rayleigh length */
00042     sunrealtyp ph0;              /**< beam center */
00043     sunrealtyp phA;              /**< beam length */
00044 };
00045
00046 /// 3D Gaussian wave structure
00047 struct gaussian3D {
00048     std::array<sunrealtyp, 3> x0;   /**< center */
00049     std::array<sunrealtyp, 3> axis; /**< direction from where it comes */
00050     sunrealtyp amp;              /**< amplitude */
00051     sunrealtyp phip;             /**< polarization rotation */
00052     sunrealtyp w0;               /**< taille */
00053     sunrealtyp zr;               /**< Rayleigh length */
00054     sunrealtyp ph0;              /**< beam center */
00055     sunrealtyp phA;              /**< beam length */
00056 };
00057
00058 /***** simulation function declarations *****/
00059
00060 /// complete 1D Simulation function
00061 void Sim1D(const std::array<sunrealtyp,2>, const int, const sunrealtyp,
00062             const sunindextyp, const bool, int *, const sunrealtyp, const int,
00063             const std::string, const int, const char,
00064             const std::vector<planewave> &,
00065             const std::vector<gaussian1D> &);
00066 /// complete 2D Simulation function
00067 void Sim2D(const std::array<sunrealtyp,2>, const int,
00068             const std::array<sunrealtyp,2>,
00069             const std::array<sunindextyp,2>, const std::array<int,2>,
00070             const bool, int *,
00071             const sunrealtyp, const int, const std::string,
00072             const int, const char,
00073             const std::vector<planewave> &, const std::vector<gaussian2D> &);
00074 /// complete 3D Simulation function
00075 void Sim3D(const std::array<sunrealtyp,2>, const int,
00076             const std::array<sunrealtyp,3>,
00077             const std::array<sunindextyp,3>, const std::array<int,3>,
00078             const bool, int *,
00079             const sunrealtyp, const int, const std::string,
00080             const int, const char,
00081             const std::vector<planewave> &, const std::vector<gaussian3D> &);
00082

```

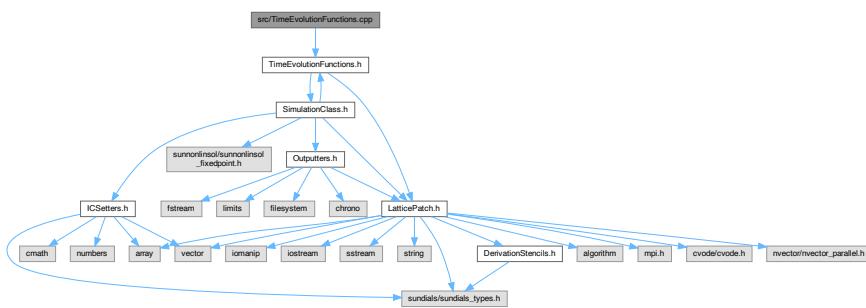
```
00083 // MPI timer function
00084 void timer(double &, double &);
```

## 6.28 src/TimeEvolutionFunctions.cpp File Reference

Implementation of functions to propagate data vectors in time according to Maxwell's equations, and various orders in the HE weak-field expansion.

```
#include "TimeEvolutionFunctions.h"
```

Include dependency graph for TimeEvolutionFunctions.cpp:



## Functions

- void [linear1DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*only under-the-hood-callable Maxwell propagation in 1D;*
- void [nonlinear1DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*nonlinear 1D HE propagation*
- void [linear2DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*only under-the-hood-callable Maxwell propagation in 2D*
- void [nonlinear2DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*nonlinear 2D HE propagation*
- void [linear3DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*only under-the-hood-callable Maxwell propagation in 3D*
- void [nonlinear3DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*nonlinear 3D HE propagation*

### 6.28.1 Detailed Description

Implementation of functions to propagate data vectors in time according to Maxwell's equations, and various orders in the HE weak-field expansion.

Definition in file [TimeEvolutionFunctions.cpp](#).

### 6.28.2 Function Documentation

### 6.28.2.1 linear1DProp()

```
void linear1DProp (
    LatticePatch * data,
    N_Vector u,
    N_Vector udot,
    int * c )
```

only under-the-hood-callable Maxwell propagation in 1D;

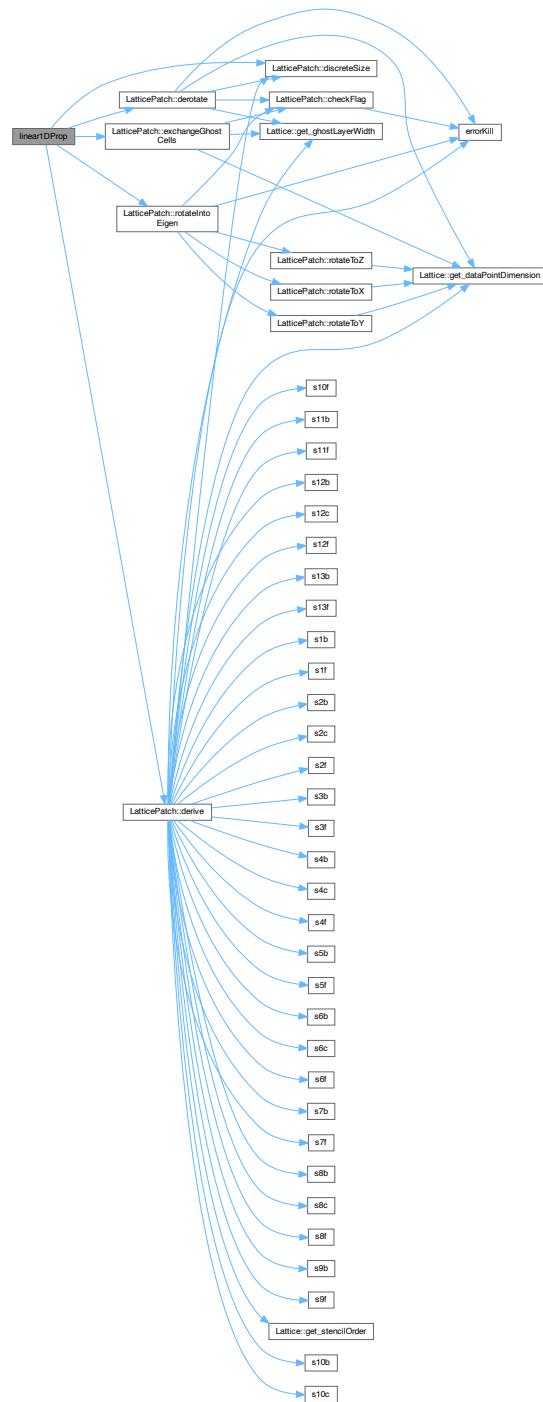
Maxwell propagation function for 1D – only for reference.

Definition at line 48 of file TimeEvolutionFunctions.cpp.

```
00048
00049
00050 // pointers to temporal and spatial derivative data
00051 sunrealtype *duData = data->duData;
00052 sunrealtype *dxData = data->buffData[1 - 1];
00053
00054 // sequence along any dimension according to the scheme:
00055 data->exchangeGhostCells(1); // -> exchange halos
00056 data->rotateIntoEigen(
00057     1); // -> rotate all data to prepare derivative operation
00058 data->derive(1); // -> perform derivative approximation operation on it
00059 data->derotate(
00060     1, dxData); // -> derotate derived data for ensuing time-evolution
00061
00062 const sunindextype totalNP = data->discreteSize();
00063 sunindextype pp = 0;
00064 for (sunindextype i = 0; i < totalNP; i++) {
00065     pp = i * 6;
00066     /*
00067         simple vacuum Maxwell equations for the temporal derivatives using the
00068         spatial derivative only in x-direction without polarization or
00069         magnetization terms
00070     */
00071     duData[pp + 0] = 0;
00072     duData[pp + 1] = -dxData[pp + 5];
00073     duData[pp + 2] = dxData[pp + 4];
00074     duData[pp + 3] = 0;
00075     duData[pp + 4] = dxData[pp + 2];
00076     duData[pp + 5] = -dxData[pp + 1];
00077 }
00078 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [LatticePatch::duData](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Here is the call graph for this function:



### 6.28.2.2 linear2DProp()

```
void linear2DProp (
    LatticePatch * data,
```

```
N_Vector u,  
N_Vector udot,  
int * c )
```

only under-the-hood-callable Maxwell propagation in 2D

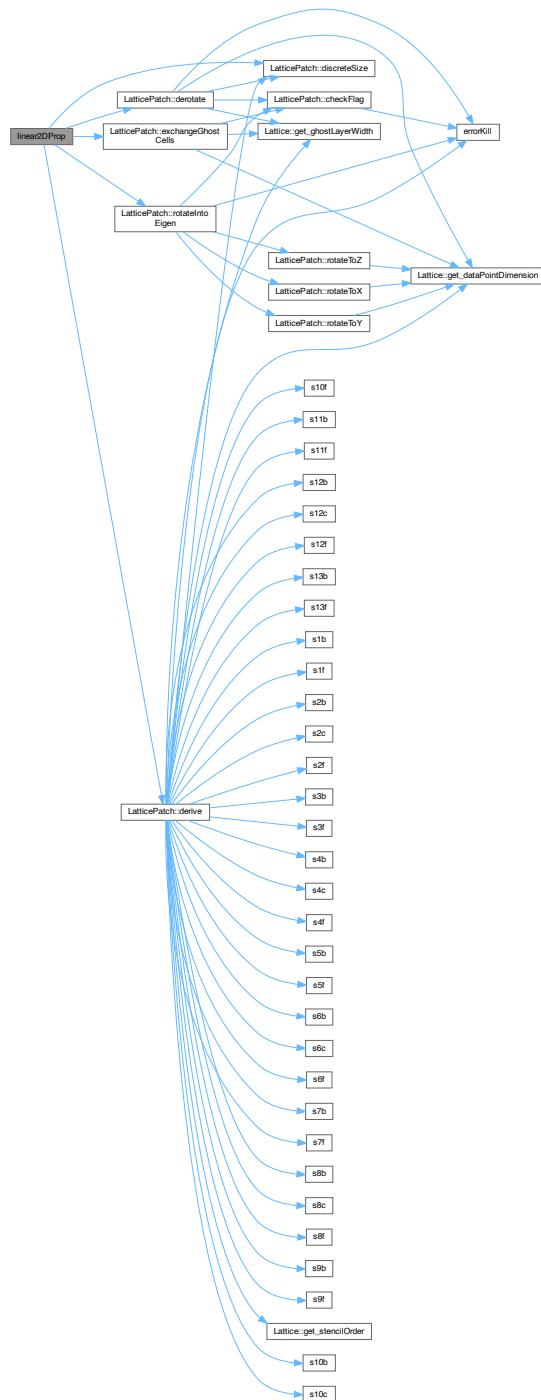
Maxwell propagation function for 2D – only for reference.

Definition at line 296 of file [TimeEvolutionFunctions.cpp](#).

```
00296  
00297  
00298     sunrealtype *duData = data->duData;  
00299     sunrealtype *dxData = data->buffData[1 - 1];  
00300     sunrealtype *dyData = data->buffData[2 - 1];  
00301  
00302     data->exchangeGhostCells(1);  
00303     data->rotateIntoEigen(1);  
00304     data->derive(1);  
00305     data->derotate(1, dxData);  
00306     data->exchangeGhostCells(2);  
00307     data->rotateIntoEigen(2);  
00308     data->derive(2);  
00309     data->derotate(2, dyData);  
00310  
00311     const sunindextype totalNP = data->discreteSize();  
00312     sunindextype pp = 0;  
00313     for (sunindextype i = 0; i < totalNP; i++) {  
00314         pp = i * 6;  
00315         duData[pp + 0] = dyData[pp + 5];  
00316         duData[pp + 1] = -dxData[pp + 5];  
00317         duData[pp + 2] = -dyData[pp + 3] + dxData[pp + 4];  
00318         duData[pp + 3] = -dyData[pp + 2];  
00319         duData[pp + 4] = dxData[pp + 2];  
00320         duData[pp + 5] = dyData[pp + 0] - dxData[pp + 1];  
00321     }  
00322 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [LatticePatch::duData](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Here is the call graph for this function:



### 6.28.2.3 linear3DProp()

```
void linear3DProp (
    LatticePatch * data,
```

```
N_Vector u,
N_Vector udot,
int * c )
```

only under-the-hood-callable Maxwell propagation in 3D

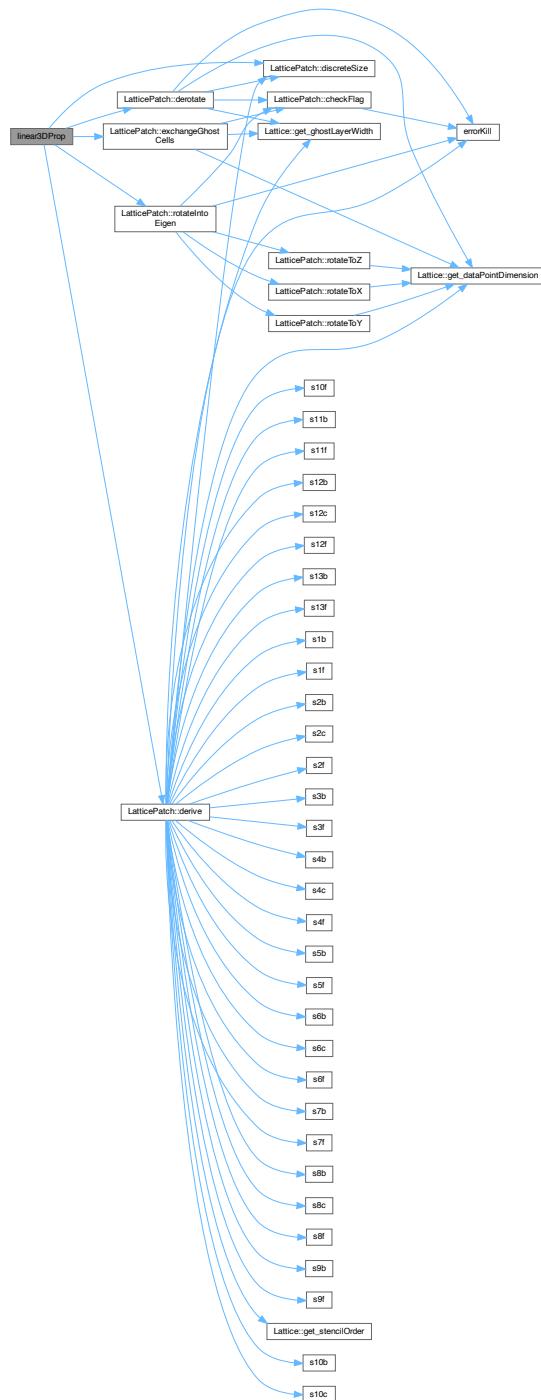
Maxwell propagation function for 3D – only for reference.

Definition at line 509 of file [TimeEvolutionFunctions.cpp](#).

```
00509
00510
00511     sunrealtype *duData = data->duData;
00512     sunrealtype *dxData = data->buffData[1 - 1];
00513     sunrealtype *dyData = data->buffData[2 - 1];
00514     sunrealtype *dzData = data->buffData[3 - 1];
00515
00516     data->exchangeGhostCells(1);
00517     data->rotateIntoEigen(1);
00518     data->derive(1);
00519     data->derotate(1, dxData);
00520     data->exchangeGhostCells(2);
00521     data->rotateIntoEigen(2);
00522     data->derive(2);
00523     data->derotate(2, dyData);
00524     data->exchangeGhostCells(3);
00525     data->rotateIntoEigen(3);
00526     data->derive(3);
00527     data->derotate(3, dzData);
00528
00529     const sunindextype totalNP = data->discreteSize();
00530     sunindextype pp = 0;
00531     for (sunindextype i = 0; i < totalNP; i++) {
00532         pp = i * 6;
00533         duData[pp + 0] = dyData[pp + 5] - dzData[pp + 4];
00534         duData[pp + 1] = dzData[pp + 3] - dxData[pp + 5];
00535         duData[pp + 2] = dxData[pp + 4] - dyData[pp + 3];
00536         duData[pp + 3] = -dyData[pp + 2] + dzData[pp + 1];
00537         duData[pp + 4] = -dzData[pp + 0] + dxData[pp + 2];
00538         duData[pp + 5] = -dxData[pp + 1] + dyData[pp + 0];
00539     }
00540 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [LatticePatch::duData](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Here is the call graph for this function:



#### 6.28.2.4 nonlinear1DProp()

```
void nonlinear1DProp (
    LatticePatch * data,
```

```
N_Vector u,
N_Vector udot,
int * c )
```

nonlinear 1D HE propagation

HE propagation function for 1D.

Definition at line 81 of file TimeEvolutionFunctions.cpp.

```
00081
00082
00083 // NVector pointers to provided field values and their temp. derivatives
00084 #if defined(_OPENMP)
00085 sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00086             *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00087 #else
00088 sunrealtype *udata = NV_DATA_P(u),
00089             *dudata = NV_DATA_P(udot);
00090 #endif
00091
00092 // pointer to spatial derivatives via pach data
00093 sunrealtype *dxData = data->buffData[1 - 1];
00094
00095 // same sequence as in the linear case
00096 data->exchangeGhostCells(1);
00097 data->rotateIntoEigen(1);
00098 data->derive(1);
00099 data->derotate(1, dxData);
00100
00101 /*
00102 F and G are nonzero in the nonlinear case,
00103 polarization and magnetization derivatives
00104 w.r.t. E- and B-field go into the e.o.m.
00105 */
00106 static sunrealtype f, g; // em field invariants F, G
00107 // derivatives of HE Lagrangian w.r.t. field invariants
00108 static sunrealtype lf, lff, lfg, lg, lgg;
00109 // matrix to hold derivatives of polarization and magnetization
00110 static std::array<sunrealtype, 21> JMM;
00111 // array to hold E^2 and B^2 components
00112 static std::array<sunrealtype, 6> Quad;
00113 // array to hold intermediate temp. derivatives of E and B
00114 static std::array<sunrealtype, 6> h;
00115 // determinant needed for explicit matrix inversion
00116 static sunrealtype detC = nan("0x12345");
00117
00118 // number of points in the patch
00119 const sunindextype totalNP = data->discreteSize();
00120 #pragma omp parallel for default(none) \
00121 private(JMM, Quad, h, detC) \
00122 shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, dxData) \
00123 schedule(static)
00124 for (sunindextype pp = 0; pp < totalNP * 6;
00125     pp += 6) { // loop over all 6dim points in the patch
00126     // em field Lorentz invariants F and G
00127     f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00128                 (Quad[1] = udata[pp + 1] * udata[pp + 1]) +
00129                 (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00130                 (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00131                 (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00132                 (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00133     g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00134         udata[pp + 2] * udata[pp + 5];
00135     // process/expansion order and corresponding derivative values of L
00136     // w.r.t. F, G
00137     switch (*c) {
00138     case 0: // linear Maxwell vacuum
00139         lf = 0;
00140         lff = 0;
00141         lfg = 0;
00142         lg = 0;
00143         lgg = 0;
00144         break;
00145     case 1: // only 4-photon processes
00146         lf = 0.000206527095658582755255648 * f;
00147         lff = 0.000206527095658582755255648;
00148         lfg = 0;
00149         lg = 0.0003614224174025198216973841 * g;
00150         lgg = 0.0003614224174025198216973841;
00151         break;
00152     case 2: // only 6-photon processes
00153         lf = 0.000354046449700427580438254 * f * f +
00154             0.000191775160254398272737387 * g * g;
```

```

00155     lff = 0.0007080928994008551608765075 * f;
00156     lfg = 0.0003835503205087965454747749 * g;
00157     lg = 0.0003835503205087965454747749 * f * g;
00158     lgg = 0.0003835503205087965454747749 * f;
00159     break;
00160 case 3: // 4- and 6-photon processes
00161     lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00162         f +
00163         0.000191775160254398272737387 * g * g;
00164     lff = 0.000206527095658582755255648 + 0.0007080928994008551608765075 * f;
00165     lfg = 0.0003835503205087965454747749 * g;
00166     lg = (0.0003614224174025198216973841 +
00167         0.0003835503205087965454747749 * f) *
00168         g;
00169     lgg = 0.0003614224174025198216973841 + 0.0003835503205087965454747749 * f;
00170     break;
00171 default:
00172     errorKill(
00173         "You need to specify a correct order in the weak-field expansion.");
00174 }
00175
00176 // derivatives of polarization and magnetization w.r.t. E and B
00177 // Jpx(Ex)
00178 JMM[0] = lf + lff * Quad[0] +
00179     udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00180 // Jpx(Ey)
00181 JMM[1] =
00182     lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00183     lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00184 // Jpy(Ey)
00185 JMM[2] = lf + lff * Quad[1] +
00186     udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00187 // Jpx(Ez) = Jpz(Ex)
00188 JMM[3] =
00189     lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00190     lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00191 // Jpy(Ez) = Jpz(Ey)
00192 JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00193     lfg * udata[2 + pp] * udata[4 + pp] +
00194     lfg * udata[1 + pp] * udata[5 + pp] +
00195     lgg * udata[4 + pp] * udata[5 + pp];
00196 // Jpz(Ez)
00197 JMM[5] = lf + lff * Quad[2] +
00198     udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00199 // Jpx(Bx) = Jmx(Ex)
00200 JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00201     (-lff + lgg) * udata[pp] * udata[3 + pp];
00202 // Jpy(Bx) = Jmx(Ey)
00203 JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00204     udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00205 // Jpz(Bx) = Jmx(Ez)
00206 JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00207     udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00208 // Jmx(Bx)
00209 JMM[9] = -lf + lgg * Quad[0] +
00210     udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00211 // Jpx(By) = Jmy(Ex)
00212 JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00213     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00214 // Jpy(By) = Jmy(Ey)
00215 JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00216     (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00217 // Jpz(By) = Jmy(Ez)
00218 JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00219     udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00220 // Jmx(By) = Jmy(Bx)
00221 JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00222     lff * udata[3 + pp] * udata[4 + pp] -
00223     lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00224 // Jmy(By)
00225 JMM[14] = -lf + lgg * Quad[1] +
00226     udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00227 // Jmz(Ex) = Jpx(Bz)
00228 JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00229     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00230 // Jmz(Ey) = Jpy(Bz)
00231 JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00232     (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00233 // Jpz(Bz) = Jmx(Ez)
00234 JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00235     (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00236 // Jmz(Bx) = Jmx(Bz)
00237 JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00238     lff * udata[3 + pp] * udata[5 + pp] -
00239     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00240 // Jmy(Bz) = Jmz(By)
00241 JMM[19] =

```

```

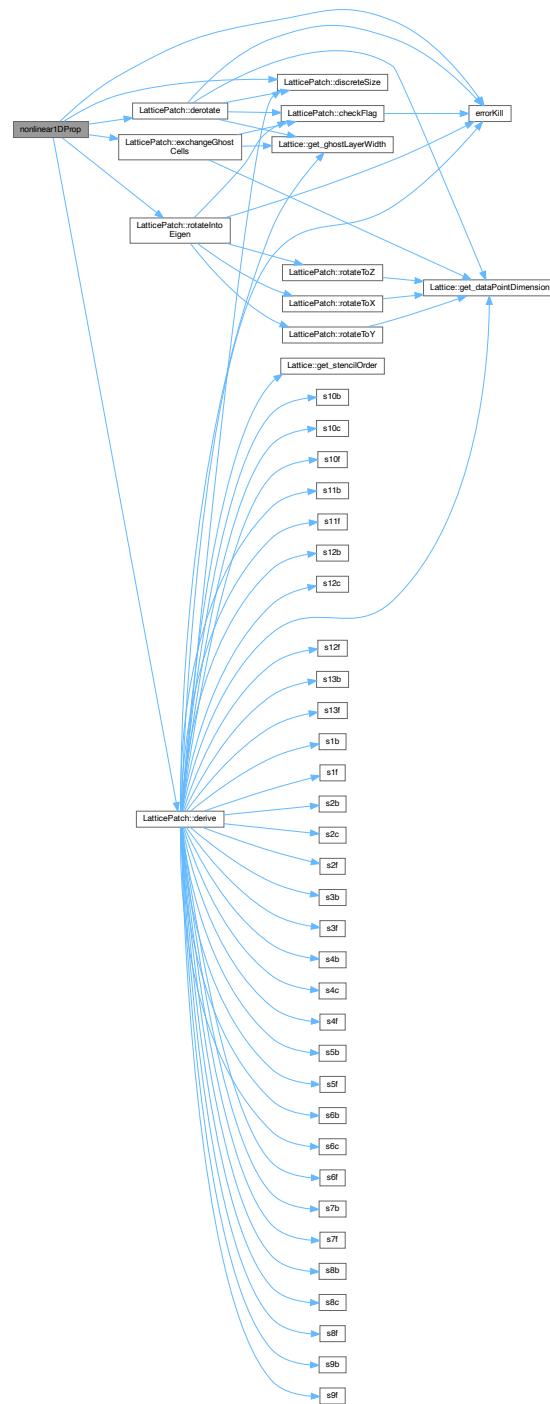
00242     lgg * udata[1 + pp] * udata[2 + pp] +
00243     lff * udata[4 + pp] * udata[5 + pp] -
00244     lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]));
00245 // Jmz(Bz)
00246 JMM[20] = -lf + lgg * Quad[2] +
00247     udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00248
00249 // apply Z
00250 // top block: -QJm(E)*E, Q-QJm(B)*B
00251 h[0] = 0;
00252 h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00253     dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00254     dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00255 h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00256     dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00257     dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00258 // bottom blocks: -Q*E
00259 h[3] = 0;
00260 h[4] = dxData[2 + pp];
00261 h[5] = -dxData[1 + pp];
00262 // (1+A)^-1 applies only to E components
00263 // -Jp(B)*B
00264 h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00265 h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00266 h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00267 // apply C^-1 explicitly, with C=1+Jp(E)
00268 dudata[pp + 0] =
00269     h[2] * (- (JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00270     h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00271     h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00272 dudata[pp + 1] =
00273     h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00274     h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00275     h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00276 dudata[pp + 2] =
00277     h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00278     h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00279     h[0] * (((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00280 detC = // determinant of C
00281     -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00282     (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00283     JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00284     JMM[1] * (- (JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00285 dudata[pp + 0] /= detC;
00286 dudata[pp + 1] /= detC;
00287 dudata[pp + 2] /= detC;
00288 dudata[pp + 3] = h[3];
00289 dudata[pp + 4] = h[4];
00290 dudata[pp + 5] = h[5];
00291 }
00292 return;
00293 }

```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Referenced by [Sim1D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.28.2.5 nonlinear2DProp()

```
void nonlinear2DProp (
    LatticePatch * data,
    N_Vector u,
    N_Vector udot,
    int * c )
```

nonlinear 2D HE propagation

HE propagation function for 2D.

Definition at line 325 of file [TimeEvolutionFunctions.cpp](#).

```
00325
00326
00327 #if defined(_OPENMP)
00328     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00329                 *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00330 #else
00331     sunrealtype *udata = NV_DATA_P(u),
00332                 *dudata = NV_DATA_P(udot);
00333 #endif
00334
00335     sunrealtype *dxData = data->buffData[1 - 1];
00336     sunrealtype *dyData = data->buffData[2 - 1];
00337
00338     data->exchangeGhostCells(1);
00339     data->rotateIntoEigen(1);
00340     data->derive(1);
00341     data->derotate(1, dxData);
00342     data->exchangeGhostCells(2);
00343     data->rotateIntoEigen(2);
00344     data->derive(2);
00345     data->derotate(2, dyData);
00346
00347     static sunrealtype f, g;
00348     static sunrealtype lf, lff, lfg, lg, lgg;
00349     static std::array<sunrealtype, 21> JMM;
00350     static std::array<sunrealtype, 6> Quad;
00351     static std::array<sunrealtype, 6> h;
00352     static sunrealtype detC;
00353
00354     const sunindextype totalNP = data->discreteSize();
00355     #pragma omp parallel for default(none) \
00356     private(JMM, Quad, h, detC) \
00357     shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, \
00358             dxData, dyData) \
00359     schedule(static)
00360     for (sunindextype pp = 0; pp < totalNP * 6; pp += 6) {
00361         f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00362                     (Quad[1] = udata[pp + 1] * udata[pp + 1]) +
00363                     (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00364                     (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00365                     (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00366                     (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00367         g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
```

```

00368     udata[pp + 2] * udata[pp + 5];
00369     switch (*c) {
00370     case 0:
00371         lff = 0;
00372         lff = 0;
00373         lfg = 0;
00374         lg = 0;
00375         lgg = 0;
00376         break;
00377     case 1:
00378         lf = 0.000206527095658582755255648 * f;
00379         lff = 0.000206527095658582755255648;
00380         lfg = 0;
00381         lg = 0.0003614224174025198216973841 * g;
00382         lgg = 0.0003614224174025198216973841;
00383         break;
00384     case 2:
00385         lf = 0.000354046449700427580438254 * f * f +
00386             0.000191775160254398272737387 * g * g;
00387         lff = 0.0007080928994008551608765075 * f;
00388         lfg = 0.0003835503205087965454747749 * g;
00389         lg = 0.0003835503205087965454747749 * f * g;
00390         lgg = 0.0003835503205087965454747749 * f;
00391         break;
00392     case 3:
00393         lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00394             f +
00395             0.000191775160254398272737387 * g * g;
00396         lff = 0.000206527095658582755255648 + 0.000708092899400855160876508 * f;
00397         lfg = 0.0003835503205087965454747749 * g;
00398         lg = (0.000361422417402519821697384 + 0.000383550320508796545474775 * f) *
00399             g;
00400         lgg = 0.000361422417402519821697384 + 0.000383550320508796545474775 * f;
00401         break;
00402     default:
00403         errorKill(
00404             "You need to specify a correct order in the weak-field expansion.");
00405     }
00406
00407 JMM[0] = lf + lff * Quad[0] +
00408     udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00409 JMM[1] =
00410     lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00411     lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00412 JMM[2] = lf + lff * Quad[1] +
00413     udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00414 JMM[3] =
00415     lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00416     lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00417 JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00418     lfg * udata[2 + pp] * udata[4 + pp] +
00419     lfg * udata[1 + pp] * udata[5 + pp] +
00420     lgg * udata[4 + pp] * udata[5 + pp];
00421 JMM[5] = lf + lff * Quad[2] +
00422     udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00423 JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00424     (-lff + lgg) * udata[pp] * udata[3 + pp];
00425 JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00426     udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00427 JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00428     udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00429 JMM[9] = -lf + lgg * Quad[0] +
00430     udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00431 JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00432     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00433 JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00434     (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00435 JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00436     udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00437 JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00438     lff * udata[3 + pp] * udata[4 + pp] -
00439     lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00440 JMM[14] = -lf + lgg * Quad[1] +
00441     udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00442 JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00443     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00444 JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00445     (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00446 JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00447     (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00448 JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00449     lff * udata[3 + pp] * udata[5 + pp] -
00450     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00451 JMM[19] =
00452     lgg * udata[1 + pp] * udata[2 + pp] +
00453     lff * udata[4 + pp] * udata[5 + pp] -
00454     lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);

```

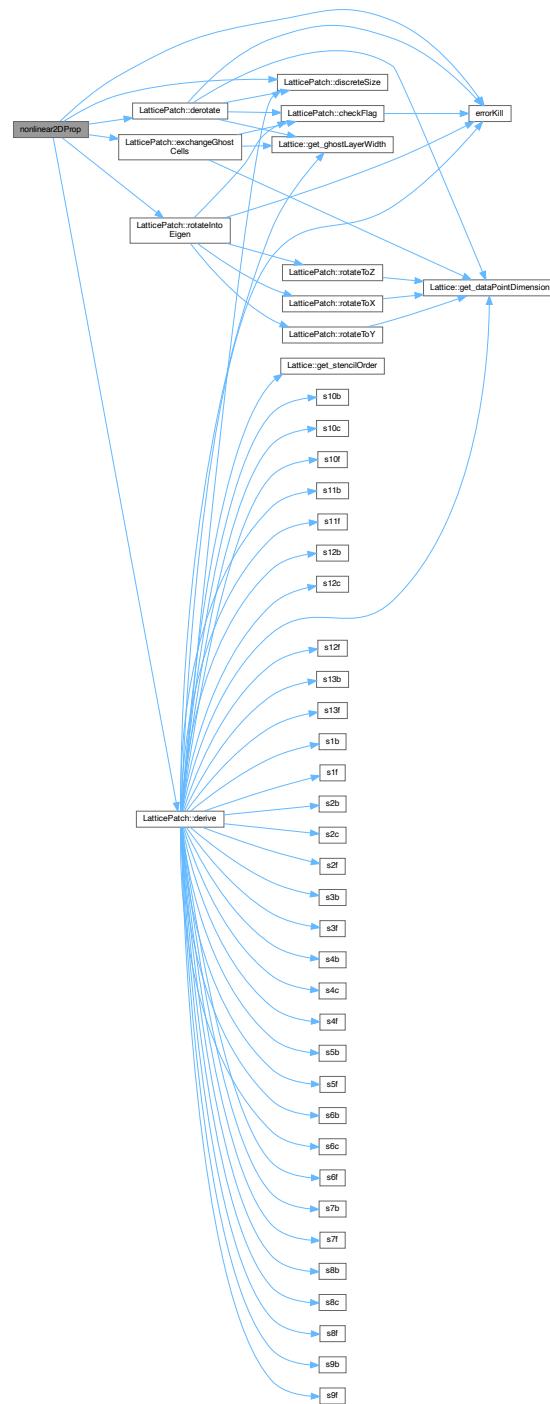
```

00455     JMM[20] = -lf + lgg * Quad[2] +
00456         udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00457
00458     h[0] = 0;
00459     h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00460         dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00461         dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00462     h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00463         dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00464         dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00465     h[3] = 0;
00466     h[4] = dxData[2 + pp];
00467     h[5] = -dxData[1 + pp];
00468     h[0] += -(dyData[pp] * JMM[15]) - dyData[1 + pp] * JMM[16] -
00469         dyData[2 + pp] * JMM[17] - dyData[3 + pp] * JMM[18] -
00470         dyData[4 + pp] * JMM[19] + dyData[5 + pp] * (1 - JMM[20]);
00471     h[1] += 0;
00472     h[2] += dyData[pp] * JMM[6] + dyData[1 + pp] * JMM[7] +
00473         dyData[2 + pp] * JMM[8] + dyData[3 + pp] * (-1 + JMM[9]) +
00474         dyData[4 + pp] * JMM[13] + dyData[5 + pp] * JMM[18];
00475     h[3] += -dyData[2 + pp];
00476     h[4] += 0;
00477     h[5] += dyData[pp];
00478     h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00479     h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00480     h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00481     dudata[pp + 0] =
00482         h[2] * (- (JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00483         h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00484         h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00485     dudata[pp + 1] =
00486         h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00487         h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00488         h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00489     dudata[pp + 2] =
00490         h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00491         h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00492         h[0] * (-((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00493     detC =
00494         -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00495         (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00496         JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00497         JMM[1] * (- (JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00498     dudata[pp + 0] /= detC;
00499     dudata[pp + 1] /= detC;
00500     dudata[pp + 2] /= detC;
00501     dudata[pp + 3] = h[3];
00502     dudata[pp + 4] = h[4];
00503     dudata[pp + 5] = h[5];
00504 }
00505 return;
00506 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Referenced by [Sim2D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.28.2.6 nonlinear3DProp()

```
void nonlinear3DProp (  
    LatticePatch * data,  
    N_Vector u,  
    N_Vector udot,  
    int * c )
```

nonlinear 3D HE propagation

## HE propagation function for 3D.

Definition at line 543 of file [TimeEvolutionFunctions.cpp](#).

```

00543
00544
00545 #if defined(_OPENMP)
00546     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00547             *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00548 #else
00549     sunrealtype *udata = NV_DATA_P(u),
00550             *dudata = NV_DATA_P(udot);
00551 #endif
00552
00553     sunrealtype *dxData = data->buffData[1 - 1];
00554     sunrealtype *dyData = data->buffData[2 - 1];
00555     sunrealtype *dzData = data->buffData[3 - 1];
00556
00557     data->exchangeGhostCells(1);
00558     data->rotateIntoEigen(1);
00559     data->derive(1);
00560     data->derotate(1,dxData);
00561     data->exchangeGhostCells(2);
00562     data->rotateIntoEigen(2);
00563     data->derive(2);
00564     data->derotate(2,dyData);
00565     data->exchangeGhostCells(3);
00566     data->rotateIntoEigen(3);
00567     data->derive(3);
00568     data->derotate(3,dzData);
00569
00570     static sunrealtype f, g;
00571     static sunrealtype lf, lff, lfg, lg, lgg;
00572     static std::array<sunrealtype, 21> JMM;
00573     static std::array<sunrealtype, 6> Quad;
00574     static std::array<sunrealtype, 6> h;
00575     static sunrealtype detC = nan("0x12345");
00576
00577     const sunindextype totalNP = data->discreteSize();
00578 #pragma omp parallel for default(none) \
00579 private(JMM, Quad, h, detC) \
00580 shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, \
00581         dxData, dyData, dzData) \
00582 schedule(static)
00583 for (sunindextype pp = 0; pp < totalNP * 6; pp += 6) {
00584     f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00585                 (Quad[1] = udata[pp + 1] * udata[pp + 1]) +

```

```

00586             (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00587             (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00588             (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00589             (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00590     g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00591         udata[pp + 2] * udata[pp + 5];
00592     switch (*c) {
00593     case 0:
00594         lf = 0;
00595         lff = 0;
00596         lfg = 0;
00597         lg = 0;
00598         lgg = 0;
00599         break;
00600     case 1:
00601         lf = 0.000206527095658582755255648 * f;
00602         lff = 0.000206527095658582755255648;
00603         lfg = 0;
00604         lg = 0.0003614224174025198216973841 * g;
00605         lgg = 0.0003614224174025198216973841;
00606         break;
00607     case 2:
00608         lf = 0.000354046449700427580438254 * f * f +
00609             0.000191775160254398272737387 * g * g;
00610         lff = 0.0007080928994008551608765075 * f;
00611         lfg = 0.0003835503205087965454747749 * g;
00612         lg = 0.0003835503205087965454747749 * f * g;
00613         lgg = 0.0003835503205087965454747749 * f;
00614         break;
00615     case 3:
00616         lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00617             f +
00618             0.000191775160254398272737387 * g * g;
00619         lff = 0.000206527095658582755255648 + 0.000708092899400855160876508 * f;
00620         lfg = 0.0003835503205087965454747749 * g;
00621         lg = (0.000361422417402519821697384 + 0.000383550320508796545474775 * f) *
00622             g;
00623         lgg = 0.000361422417402519821697384 + 0.000383550320508796545474775 * f;
00624         break;
00625     default:
00626         errorKill(
00627             "You need to specify a correct order in the weak-field expansion.");
00628     }
00629
00630     JMM[0] = lf + lff * Quad[0] +
00631         udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00632     JMM[1] =
00633         lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00634         lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00635     JMM[2] = lf + lff * Quad[1] +
00636         udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00637     JMM[3] =
00638         lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00639         lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00640     JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00641             lfg * udata[2 + pp] * udata[4 + pp] +
00642             lfg * udata[1 + pp] * udata[5 + pp] +
00643             lgg * udata[4 + pp] * udata[5 + pp];
00644     JMM[5] = lf + lff * Quad[2] +
00645         udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00646     JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00647             (-lff + lgg) * udata[pp] * udata[3 + pp];
00648     JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00649             udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00650     JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00651             udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00652     JMM[9] = -lf + lgg * Quad[0] +
00653         udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00654     JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00655             (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00656     JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00657             (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00658     JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00659             udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00660     JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00661         lff * udata[3 + pp] * udata[4 + pp] -
00662         lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00663     JMM[14] = -lf + lgg * Quad[1] +
00664         udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00665     JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00666             (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00667     JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00668             (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00669     JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00670             (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00671     JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00672             lff * udata[3 + pp] * udata[5 + pp] -

```

```

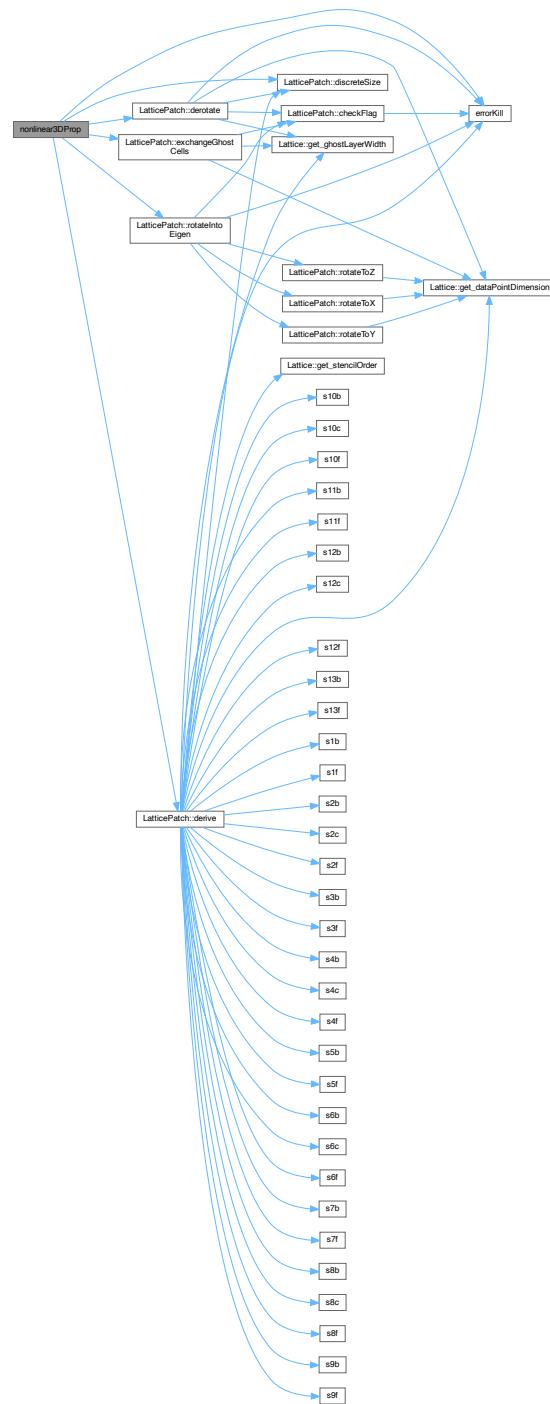
00673     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00674     JMM[19] =
00675         lgg * udata[1 + pp] * udata[2 + pp] +
00676         lff * udata[4 + pp] * udata[5 + pp] -
00677         lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);
00678     JMM[20] = -lf + lgg * Quad[2] +
00679         udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00680
00681     h[0] = 0;
00682     h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00683         dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00684         dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00685     h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00686         dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00687         dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00688     h[3] = 0;
00689     h[4] = dxData[2 + pp];
00690     h[5] = -dxData[1 + pp];
00691     h[0] += -(dyData[pp] * JMM[15]) - dyData[1 + pp] * JMM[16] -
00692         dyData[2 + pp] * JMM[17] - dyData[3 + pp] * JMM[18] -
00693         dyData[4 + pp] * JMM[19] + dyData[5 + pp] * (1 - JMM[20]);
00694     h[1] += 0;
00695     h[2] += dyData[pp] * JMM[6] + dyData[1 + pp] * JMM[7] +
00696         dyData[2 + pp] * JMM[8] + dyData[3 + pp] * (-1 + JMM[9]) +
00697         dyData[4 + pp] * JMM[13] + dyData[5 + pp] * JMM[18];
00698     h[3] += -dyData[2 + pp];
00699     h[4] += 0;
00700     h[5] += dyData[pp];
00701     h[0] += dzData[pp] * JMM[10] + dzData[1 + pp] * JMM[11] +
00702         dzData[2 + pp] * JMM[12] + dzData[3 + pp] * JMM[13] +
00703         dzData[4 + pp] * (-1 + JMM[14]) + dzData[5 + pp] * JMM[19];
00704     h[1] += -(dzData[pp] * JMM[6]) - dzData[1 + pp] * JMM[7] -
00705         dzData[2 + pp] * JMM[8] + dzData[3 + pp] * (1 - JMM[9]) -
00706         dzData[4 + pp] * JMM[13] - dzData[5 + pp] * JMM[18];
00707     h[2] += 0;
00708     h[3] += dzData[1 + pp];
00709     h[4] += -dzData[pp];
00710     h[5] += 0;
00711     h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00712     h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00713     h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00714     dudata[pp + 0] =
00715         h[2] * (-JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00716         h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00717         h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00718     dudata[pp + 1] =
00719         h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00720         h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00721         h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00722     dudata[pp + 2] =
00723         h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00724         h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00725         h[0] * (-(1 + JMM[2]) * JMM[3] + JMM[1] * JMM[4]);
00726     detC =
00727         -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00728         (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00729         JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00730         JMM[1] * (-(JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00731     dudata[pp + 0] /= detC;
00732     dudata[pp + 1] /= detC;
00733     dudata[pp + 2] /= detC;
00734     dudata[pp + 3] = h[3];
00735     dudata[pp + 4] = h[4];
00736     dudata[pp + 5] = h[5];
00737 }
00738 return;
00739 }

```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Referenced by [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.29 TimeEvolutionFunctions.cpp

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file TimeEvolutionFunctions.cpp
00003 /// @brief Implementation of functions to propagate
00004 /// data vectors in time according to Maxwell's equations,
00005 /// and various orders in the HE weak-field expansion
00006 ///////////////////////////////////////////////////////////////////
00007
00008 #include "TimeEvolutionFunctions.h"
00009
00010 /// CCode right-hand-side function (CVRhsFn)
00011 int TimeEvolution::f(sunrealtype t, N_Vector u, N_Vector udot, void *data_loc) {
00012
00013     // Set recover pointer to provided lattice patch where the field data resides
00014     LatticePatch *data = static_cast<LatticePatch *>(data_loc);
00015
00016     // update circle
00017     // Access provided field values and temp. derivatievees with NVector pointers
00018 #if defined(_OPENMP)
00019     unrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00020                 *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00021 #else
00022     unrealtype *udata = NV_DATA_P(u),
00023                 *dudata = NV_DATA_P(udot);
00024 #endif
00025
00026     // Store original data location of the patch
00027     unrealtype *originaluData = data->uData,
00028                 *originalduData = data->duData;
00029
00030     // Point patch data to arguments of f
00031     data->uData = udata;
00032     data->duData = dudata;
00033
00034     // Time-evolve these arguments (the field data) with specific propagator below
00035     TimeEvolver(data, u, udot, c);
00036
00037     // Refer patch data back to original location
00038     data->uData = originaluData;
00039     data->duData = originalduData;
00040
00041     return (0);
00042 }
00043
00044 /// only under-the-hood-callable Maxwell propagation in 1D;
00045 // unused parameters 2-4 for compliance with CVRhsFn - field data is here
00046 // accessed implicitly via user data (lattice patch);
00047 // same effect as the respective nonlinear function without nonlinear terms
00048 void linear1DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c) {
00049
00050     // pointers to temporal and spatial derivative data
00051     unrealtype *duData = data->duData;
00052     unrealtype *dxData = data->buffData[1 - 1];
00053
00054     // sequence along any dimension according to the scheme:
00055     data->exchangeGhostCells(1); // -> exchange halos
00056     data->rotateIntoEigen(
00057         1); // -> rotate all data to prepare derivative operation
00058     data->derive(1); // -> perform derivative approximation operation on it
00059     data->derotate(
00060         1, dxData); // -> derotate derived data for ensuing time-evolution
00061

```

```

00062 const sunindextype totalNP = data->discreteSize();
00063 sunindextype pp = 0;
00064 for (sunindextype i = 0; i < totalNP; i++) {
00065     pp = i * 6;
00066     /*
00067         simple vacuum Maxwell equations for the temporal derivatives using the
00068         spatial derivative only in x-direction without polarization or
00069         magnetization terms
00070     */
00071     duData[pp + 0] = 0;
00072     duData[pp + 1] = -dxData[pp + 5];
00073     duData[pp + 2] = dxData[pp + 4];
00074     duData[pp + 3] = 0;
00075     duData[pp + 4] = dxData[pp + 2];
00076     duData[pp + 5] = -dxData[pp + 1];
00077 }
00078 }
00079
00080 /// nonlinear 1D HE propagation
00081 void nonlinear1DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c) {
00082
00083     // NVector pointers to provided field values and their temp. derivatives
00084 #if defined(_OPENMP)
00085     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00086                 *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00087 #else
00088     sunrealtype *udata = NV_DATA_P(u),
00089                 *dudata = NV_DATA_P(udot);
00090 #endif
00091
00092     // pointer to spatial derivatives via patch data
00093     sunrealtype *dxData = data->buffData[1 - 1];
00094
00095     // same sequence as in the linear case
00096     data->exchangeGhostCells(1);
00097     data->rotateIntoEigen(1);
00098     data->derive(1);
00099     data->derotate(1, dxData);
00100
00101     /*
00102     F and G are nonzero in the nonlinear case,
00103     polarization and magnetization derivatives
00104     w.r.t. E- and B-field go into the e.o.m.
00105     */
00106     static sunrealtype f, g; // em field invariants F, G
00107     // derivatives of HE Lagrangian w.r.t. field invariants
00108     static sunrealtype lf, lff, lfg, lg, lgg;
00109     // matrix to hold derivatives of polarization and magnetization
00110     static std::array<sunrealtype, 21> JMM;
00111     // array to hold E^2 and B^2 components
00112     static std::array<sunrealtype, 6> Quad;
00113     // array to hold intermediate temp. derivatives of E and B
00114     static std::array<sunrealtype, 6> h;
00115     // determinant needed for explicit matrix inversion
00116     static sunrealtype detC = nan("0x12345");
00117
00118     // number of points in the patch
00119     const sunindextype totalNP = data->discreteSize();
00120     #pragma omp parallel for default(none) \
00121     private(JMM, Quad, h, detC) \
00122     shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, dxData) \
00123     schedule(static)
00124     for (sunindextype pp = 0; pp < totalNP * 6;
00125          pp += 6) { // loop over all 6dim points in the patch
00126         // em field Lorentz invariants F and G
00127         f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00128                     (Quad[1] = udata[pp + 1] * udata[pp + 1]) +
00129                     (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00130                     (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00131                     (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00132                     (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00133         g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00134             udata[pp + 2] * udata[pp + 5];
00135         // process/expansion order and corresponding derivative values of L
00136         // w.r.t. F, G
00137         switch (*c) {
00138             case 0: // linear Maxwell vacuum
00139                 lf = 0;
00140                 lff = 0;
00141                 lfg = 0;
00142                 lg = 0;
00143                 lgg = 0;
00144                 break;
00145             case 1: // only 4-photon processes
00146                 lf = 0.000206527095658582755255648 * f;
00147                 lff = 0.000206527095658582755255648;
00148                 lfg = 0;

```

```

00149     lg = 0.0003614224174025198216973841 * g;
00150     lgg = 0.0003614224174025198216973841;
00151     break;
00152 case 2: // only 6-photon processes
00153     lf = 0.000354046449700427580438254 * f * f +
00154         0.000191775160254398272737387 * g * g;
00155     lff = 0.0007080928994008551608765075 * f;
00156     lfg = 0.0003835503205087965454747749 * g;
00157     lg = 0.0003835503205087965454747749 * f * g;
00158     lgg = 0.0003835503205087965454747749 * f;
00159     break;
00160 case 3: // 4- and 6-photon processes
00161     lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00162         f +
00163         0.000191775160254398272737387 * g * g;
00164     lff = 0.000206527095658582755255648 + 0.0007080928994008551608765075 * f;
00165     lfg = 0.0003835503205087965454747749 * g;
00166     lg = (0.0003614224174025198216973841 +
00167         0.0003835503205087965454747749 * f) *
00168         g;
00169     lgg = 0.0003614224174025198216973841 + 0.0003835503205087965454747749 * f;
00170     break;
00171 default:
00172     errorKill(
00173         "You need to specify a correct order in the weak-field expansion.");
00174 }
00175
00176 // derivatives of polarization and magnetization w.r.t. E and B
00177 // Jpx(Ex)
00178 JMM[0] = lf + lff * Quad[0] +
00179     udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00180 // Jpx(Ey)
00181 JMM[1] =
00182     lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00183     lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00184 // Jpy(Ey)
00185 JMM[2] = lf + lff * Quad[1] +
00186     udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00187 // Jpx(Ez) = Jpz(Ex)
00188 JMM[3] =
00189     lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00190     lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00191 // Jpy(Ez) = Jpz(Ey)
00192 JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00193     lfg * udata[2 + pp] * udata[4 + pp] +
00194     lfg * udata[1 + pp] * udata[5 + pp] +
00195     lgg * udata[4 + pp] * udata[5 + pp];
00196 // Jpz(Ez)
00197 JMM[5] = lf + lff * Quad[2] +
00198     udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00199 // Jpx(Bx) = Jmx(Ex)
00200 JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00201     (-lff + lgg) * udata[pp] * udata[3 + pp];
00202 // Jpy(Bx) = Jmx(Ey)
00203 JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00204     udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00205 // Jpz(Bx) = Jmx(Ez)
00206 JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00207     udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00208 // Jmx(Bx)
00209 JMM[9] = -lf + lgg * Quad[0] +
00210     udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00211 // Jpx(By) = Jmy(Ex)
00212 JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00213     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00214 // Jpy(By) = Jmy(Ey)
00215 JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00216     (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00217 // Jpz(By) = Jmy(Ez)
00218 JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00219     udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00220 // Jmx(By) = Jmy(Bx)
00221 JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00222     lff * udata[3 + pp] * udata[4 + pp] -
00223     lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00224 // Jmy(By)
00225 JMM[14] = -lf + lgg * Quad[1] +
00226     udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00227 // Jmz(Ex) = Jpx(Bz)
00228 JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00229     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00230 // Jmz(Ey) = Jpy(Bz)
00231 JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00232     (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00233 // Jpz(Bz) = Jmz(Ez)
00234 JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00235     (-lff + lgg) * udata[2 + pp] * udata[5 + pp];

```

```

00236 // Jmz(Bx) = Jmx(Bz)
00237 JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00238     lff * udata[3 + pp] * udata[5 + pp] -
00239     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00240 // Jmy(Bz) = Jmz(By)
00241 JMM[19] =
00242     lgg * udata[1 + pp] * udata[2 + pp] +
00243     lff * udata[4 + pp] * udata[5 + pp] -
00244     lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);
00245 // Jmz(Bz)
00246 JMM[20] = -lf + lgg * Quad[2] +
00247     udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00248
00249 // apply Z
00250 // top block: -QJm(E)*E, Q-QJm(B)*B
00251 h[0] = 0;
00252 h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00253     dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00254     dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00255 h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00256     dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00257     dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00258 // bottom blocks: -Q*E
00259 h[3] = 0;
00260 h[4] = dxData[2 + pp];
00261 h[5] = -dxData[1 + pp];
00262 // (1+A)^-1 applies only to E components
00263 // -Jp(B)*B
00264 h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00265 h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00266 h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00267 // apply C^-1 explicitly, with C=1+Jp(E)
00268 dudata[pp + 0] =
00269     h[2] * (-JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00270     h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00271     h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00272 dudata[pp + 1] =
00273     h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00274     h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00275     h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00276 dudata[pp + 2] =
00277     h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00278     h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00279     h[0] * ((-((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00280 detC = // determinant of C
00281     -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00282     (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00283     JMM[0] * (1 + JMM[2]) - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00284     JMM[1] * (-JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00285 dudata[pp + 0] /= detC;
00286 dudata[pp + 1] /= detC;
00287 dudata[pp + 2] /= detC;
00288 dudata[pp + 3] = h[3];
00289 dudata[pp + 4] = h[4];
00290 dudata[pp + 5] = h[5];
00291 }
00292 return;
00293 }
00294
00295 /// only under-the-hood-callable Maxwell propagation in 2D
00296 void linear2DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c) {
00297
00298     sunrealtype *duData = data->duData;
00299     sunrealtype *dxData = data->buffData[1 - 1];
00300     sunrealtype *dyData = data->buffData[2 - 1];
00301
00302     data->exchangeGhostCells(1);
00303     data->rotateIntoEigen(1);
00304     data->derive(1);
00305     data->derotate(1, dxData);
00306     data->exchangeGhostCells(2);
00307     data->rotateIntoEigen(2);
00308     data->derive(2);
00309     data->derotate(2, dyData);
00310
00311     const sunindextype totalNP = data->discreteSize();
00312     sunindextype pp = 0;
00313     for (sunindextype i = 0; i < totalNP; i++) {
00314         pp = i * 6;
00315         duData[pp + 0] = dyData[pp + 5];
00316         duData[pp + 1] = -dxData[pp + 5];
00317         duData[pp + 2] = -dyData[pp + 3] + dxData[pp + 4];
00318         duData[pp + 3] = -dyData[pp + 2];
00319         duData[pp + 4] = dxData[pp + 2];
00320         duData[pp + 5] = dyData[pp + 0] - dxData[pp + 1];
00321     }
00322 }

```

```

00323
00324 // non-linear 2D HE propagation
00325 void nonlinear2DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c) {
00326
00327 #if defined(_OPENMP)
00328     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00329                 *udata = N_VGetArrayPointer_MPIPlusX(udot);
00330 #else
00331     sunrealtype *udata = NV_DATA_P(u),
00332                 *udata = NV_DATA_P(udot);
00333 #endif
00334
00335     sunrealtype *dxData = data->buffData[1 - 1];
00336     sunrealtype *dyData = data->buffData[2 - 1];
00337
00338     data->exchangeGhostCells(1);
00339     data->rotateIntoEigen(1);
00340     data->derive(1);
00341     data->derotate(1, dxData);
00342     data->exchangeGhostCells(2);
00343     data->rotateIntoEigen(2);
00344     data->derive(2);
00345     data->derotate(2, dyData);
00346
00347     static sunrealtype f, g;
00348     static sunrealtype lf, lff, lfg, lg, lgg;
00349     static std::array<sunrealtype, 21> JMM;
00350     static std::array<sunrealtype, 6> Quad;
00351     static std::array<sunrealtype, 6> h;
00352     static sunrealtype detC;
00353
00354     const sunindextype totalNP = data->discreteSize();
00355 #pragma omp parallel for default(none) \
00356 private(JMM, Quad, h, detC) \
00357 shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, \
00358         dxData, dyData) \
00359 schedule(static)
00360     for (sunindextype pp = 0; pp < totalNP * 6; pp += 6) {
00361         f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00362                     (Quad[1] = udata[pp + 1] * udata[pp + 1]) +
00363                     (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00364                     (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00365                     (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00366                     (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00367         g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00368             udata[pp + 2] * udata[pp + 5];
00369         switch (*c) {
00370             case 0:
00371                 lf = 0;
00372                 lff = 0;
00373                 lfg = 0;
00374                 lg = 0;
00375                 lgg = 0;
00376                 break;
00377             case 1:
00378                 lf = 0.000206527095658582755255648 * f;
00379                 lff = 0.000206527095658582755255648;
00380                 lfg = 0;
00381                 lg = 0.0003614224174025198216973841 * g;
00382                 lgg = 0.0003614224174025198216973841;
00383                 break;
00384             case 2:
00385                 lf = 0.000354046449700427580438254 * f * f +
00386                     0.000191775160254398272737387 * g * g;
00387                 lff = 0.0007080928994008551608765075 * f;
00388                 lfg = 0.0003835503205087965454747749 * g;
00389                 lg = 0.0003835503205087965454747749 * f * g;
00390                 lgg = 0.0003835503205087965454747749 * f;
00391                 break;
00392             case 3:
00393                 lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00394                     f +
00395                     0.000191775160254398272737387 * g * g;
00396                 lff = 0.000206527095658582755255648 + 0.000708092899400855160876508 * f;
00397                 lfg = 0.0003835503205087965454747749 * g;
00398                 lg = (0.000361422417402519821697384 + 0.000383550320508796545474775 * f) *
00399                     g;
00400                 lgg = 0.000361422417402519821697384 + 0.000383550320508796545474775 * f;
00401                 break;
00402             default:
00403                 errorKill(
00404                     "You need to specify a correct order in the weak-field expansion.");
00405             }
00406
00407             JMM[0] = lf + lff * Quad[0] +
00408                 udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00409             JMM[1] =

```

```

00410     lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00411     lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00412     JMM[2] = lf + lff * Quad[1] +
00413     udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00414     JMM[3] =
00415     lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00416     lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00417     JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00418     lfg * udata[2 + pp] * udata[4 + pp] +
00419     lfg * udata[1 + pp] * udata[5 + pp] +
00420     lgg * udata[4 + pp] * udata[5 + pp];
00421     JMM[5] = lf + lff * Quad[2] +
00422     udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00423     JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00424     (-lff + lgg) * udata[pp] * udata[3 + pp];
00425     JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00426     udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00427     JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00428     udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00429     JMM[9] = -lf + lgg * Quad[0] +
00430     udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00431     JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00432     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00433     JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00434     (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00435     JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00436     udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00437     JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00438     lff * udata[3 + pp] * udata[4 + pp] -
00439     lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00440     JMM[14] = -lf + lgg * Quad[1] +
00441     udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00442     JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00443     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00444     JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00445     (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00446     JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00447     (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00448     JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00449     lff * udata[3 + pp] * udata[5 + pp] -
00450     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00451     JMM[19] =
00452     lgg * udata[1 + pp] * udata[2 + pp] +
00453     lff * udata[4 + pp] * udata[5 + pp] -
00454     lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);
00455     JMM[20] = -lf + lgg * Quad[2] +
00456     udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00457
00458     h[0] = 0;
00459     h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00460     dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00461     dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00462     h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00463     dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00464     dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00465     h[3] = 0;
00466     h[4] = dxData[2 + pp];
00467     h[5] = -dxData[1 + pp];
00468     h[0] += -(dyData[pp] * JMM[15]) - dyData[1 + pp] * JMM[16] -
00469     dyData[2 + pp] * JMM[17] - dyData[3 + pp] * JMM[18] -
00470     dyData[4 + pp] * JMM[19] + dyData[5 + pp] * (1 - JMM[20]);
00471     h[1] += 0;
00472     h[2] += dyData[pp] * JMM[6] + dyData[1 + pp] * JMM[7] +
00473     dyData[2 + pp] * JMM[8] + dyData[3 + pp] * (-1 + JMM[9]) +
00474     dyData[4 + pp] * JMM[13] + dyData[5 + pp] * JMM[18];
00475     h[3] += -dyData[2 + pp];
00476     h[4] += 0;
00477     h[5] += dyData[pp];
00478     h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00479     h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00480     h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00481     dudata[pp + 0] =
00482     h[2] * (-JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00483     h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00484     h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00485     dudata[pp + 1] =
00486     h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00487     h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00488     h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00489     dudata[pp + 2] =
00490     h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00491     h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00492     h[0] * (((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00493     detC =
00494     -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00495     (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00496     JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -

```

```

00497     JMM[1] * (- (JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00498     dudata[pp + 0] /= detC;
00499     dudata[pp + 1] /= detC;
00500     dudata[pp + 2] /= detC;
00501     dudata[pp + 3] = h[3];
00502     dudata[pp + 4] = h[4];
00503     dudata[pp + 5] = h[5];
00504 }
00505 return;
00506 }
00507
00508 // only under-the-hood-callable Maxwell propagation in 3D
00509 void linear3DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c) {
00510
00511     sunrealtype *duData = data->duData;
00512     sunrealtype *dxData = data->buffData[1 - 1];
00513     sunrealtype *dyData = data->buffData[2 - 1];
00514     sunrealtype *dzData = data->buffData[3 - 1];
00515
00516     data->exchangeGhostCells(1);
00517     data->rotateIntoEigen(1);
00518     data->derive(1);
00519     data->derotate(1, dxData);
00520     data->exchangeGhostCells(2);
00521     data->rotateIntoEigen(2);
00522     data->derive(2);
00523     data->derotate(2, dyData);
00524     data->exchangeGhostCells(3);
00525     data->rotateIntoEigen(3);
00526     data->derive(3);
00527     data->derotate(3, dzData);
00528
00529     const sunindextype totalNP = data->discreteSize();
00530     sunindextype pp = 0;
00531     for (sunindextype i = 0; i < totalNP; i++) {
00532         pp = i * 6;
00533         duData[pp + 0] = dyData[pp + 5] - dzData[pp + 4];
00534         duData[pp + 1] = dzData[pp + 3] - dxData[pp + 5];
00535         duData[pp + 2] = dxData[pp + 4] - dyData[pp + 3];
00536         duData[pp + 3] = -dyData[pp + 2] + dzData[pp + 1];
00537         duData[pp + 4] = -dzData[pp + 0] + dxData[pp + 2];
00538         duData[pp + 5] = -dxData[pp + 1] + dyData[pp + 0];
00539     }
00540 }
00541
00542 // nonlinear 3D HE propagation
00543 void nonlinear3DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c) {
00544
00545 #if defined(_OPENMP)
00546     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00547                 *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00548 #else
00549     sunrealtype *udata = NV_DATA_P(u),
00550                 *dudata = NV_DATA_P(udot);
00551 #endif
00552
00553     sunrealtype *dxData = data->buffData[1 - 1];
00554     sunrealtype *dyData = data->buffData[2 - 1];
00555     sunrealtype *dzData = data->buffData[3 - 1];
00556
00557     data->exchangeGhostCells(1);
00558     data->rotateIntoEigen(1);
00559     data->derive(1);
00560     data->derotate(1, dxData);
00561     data->exchangeGhostCells(2);
00562     data->rotateIntoEigen(2);
00563     data->derive(2);
00564     data->derotate(2, dyData);
00565     data->exchangeGhostCells(3);
00566     data->rotateIntoEigen(3);
00567     data->derive(3);
00568     data->derotate(3, dzData);
00569
00570     static sunrealtype f, g;
00571     static sunrealtype lf, lff, lfg, lg, lgg;
00572     static std::array<sunrealtype, 21> JMM;
00573     static std::array<sunrealtype, 6> Quad;
00574     static std::array<sunrealtype, 6> h;
00575     static sunrealtype detC = nan("0x12345");
00576
00577     const sunindextype totalNP = data->discreteSize();
00578 #pragma omp parallel for default(none) \
00579 private(JMM, Quad, h, detC) \
00580 shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, \
00581           dxData, dyData, dzData) \
00582 schedule(static)
00583     for (sunindextype pp = 0; pp < totalNP * 6; pp += 6) {

```

```

00584     f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00585             (Quad[1] = udata[pp + 1] * udata[pp + 1]) +
00586             (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00587             (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00588             (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00589             (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00590     g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00591         udata[pp + 2] * udata[pp + 5];
00592     switch (*c) {
00593     case 0:
00594         lf = 0;
00595         lff = 0;
00596         lfg = 0;
00597         lg = 0;
00598         lgg = 0;
00599         break;
00600     case 1:
00601         lf = 0.000206527095658582755255648 * f;
00602         lff = 0.000206527095658582755255648;
00603         lfg = 0;
00604         lg = 0.0003614224174025198216973841 * g;
00605         lgg = 0.0003614224174025198216973841;
00606         break;
00607     case 2:
00608         lf = 0.000354046449700427580438254 * f * f +
00609             0.000191775160254398272737387 * g * g;
00610         lff = 0.0007080928994008551608765075 * f;
00611         lfg = 0.0003835503205087965454747749 * g;
00612         lg = 0.0003835503205087965454747749 * f * g;
00613         lgg = 0.0003835503205087965454747749 * f;
00614         break;
00615     case 3:
00616         lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00617             f +
00618             0.000191775160254398272737387 * g * g;
00619         lff = 0.000206527095658582755255648 + 0.000708092899400855160876508 * f;
00620         lfg = 0.0003835503205087965454747749 * g;
00621         lg = (0.000361422417402519821697384 + 0.000383550320508796545474775 * f) *
00622             g;
00623         lgg = 0.000361422417402519821697384 + 0.000383550320508796545474775 * f;
00624         break;
00625     default:
00626         errorKill(
00627             "You need to specify a correct order in the weak-field expansion.");
00628     }
00629
00630     JMM[0] = lf + lff * Quad[0] +
00631         udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00632     JMM[1] =
00633         lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00634         lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00635     JMM[2] = lf + lff * Quad[1] +
00636         udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00637     JMM[3] =
00638         lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00639         lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00640     JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00641         lfg * udata[2 + pp] * udata[4 + pp] +
00642         lfg * udata[1 + pp] * udata[5 + pp] +
00643         lgg * udata[4 + pp] * udata[5 + pp];
00644     JMM[5] = lf + lff * Quad[2] +
00645         udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00646     JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00647         (-lff + lgg) * udata[pp] * udata[3 + pp];
00648     JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00649         udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00650     JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00651         udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00652     JMM[9] = -lf + lgg * Quad[0] +
00653         udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00654     JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00655         (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00656     JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00657         (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00658     JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00659         udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00660     JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00661         lff * udata[3 + pp] * udata[4 + pp] -
00662         lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00663     JMM[14] = -lf + lgg * Quad[1] +
00664         udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00665     JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00666         (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00667     JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00668         (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00669     JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00670         (-lff + lgg) * udata[2 + pp] * udata[5 + pp];

```

```

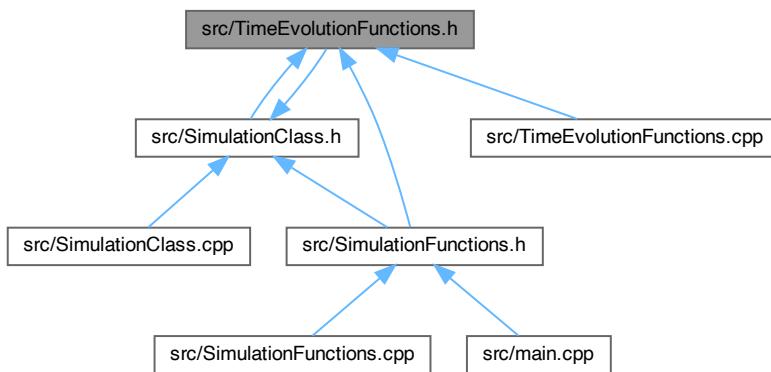
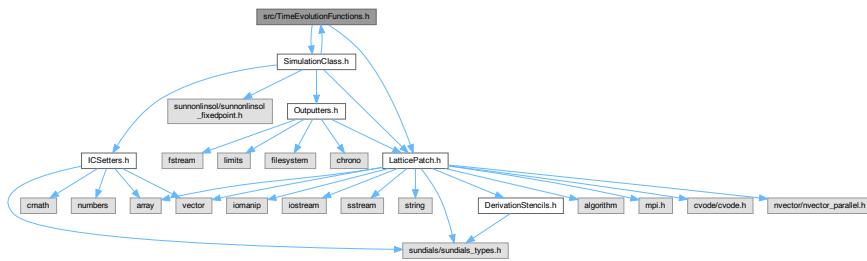
00671     JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00672         lff * udata[3 + pp] * udata[5 + pp] -
00673         lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00674     JMM[19] =
00675         lgg * udata[1 + pp] * udata[2 + pp] +
00676         lff * udata[4 + pp] * udata[5 + pp] -
00677         lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);
00678     JMM[20] = -lff + lgg * Quad[2] +
00679         udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00680
00681     h[0] = 0;
00682     h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00683         dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00684         dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00685     h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00686         dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00687         dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00688     h[3] = 0;
00689     h[4] = dxData[2 + pp];
00690     h[5] = -dxData[1 + pp];
00691     h[0] += -(dyData[pp] * JMM[15]) - dyData[1 + pp] * JMM[16] -
00692         dyData[2 + pp] * JMM[17] - dyData[3 + pp] * JMM[18] -
00693         dyData[4 + pp] * JMM[19] + dyData[5 + pp] * (1 - JMM[20]);
00694     h[1] += 0;
00695     h[2] += dyData[pp] * JMM[6] + dyData[1 + pp] * JMM[7] +
00696         dyData[2 + pp] * JMM[8] + dyData[3 + pp] * (-1 + JMM[9]) +
00697         dyData[4 + pp] * JMM[13] + dyData[5 + pp] * JMM[18];
00698     h[3] += -dyData[2 + pp];
00699     h[4] += 0;
00700     h[5] += dyData[pp];
00701     h[0] += dzData[pp] * JMM[10] + dzData[1 + pp] * JMM[11] +
00702         dzData[2 + pp] * JMM[12] + dzData[3 + pp] * JMM[13] +
00703         dzData[4 + pp] * (-1 + JMM[14]) + dzData[5 + pp] * JMM[19];
00704     h[1] += -(dzData[pp] * JMM[6]) - dzData[1 + pp] * JMM[7] -
00705         dzData[2 + pp] * JMM[8] + dzData[3 + pp] * (1 - JMM[9]) -
00706         dzData[4 + pp] * JMM[13] - dzData[5 + pp] * JMM[18];
00707     h[2] += 0;
00708     h[3] += dzData[1 + pp];
00709     h[4] += -dzData[pp];
00710     h[5] += 0;
00711     h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00712     h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00713     h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00714     udata[pp + 0] =
00715         h[2] * (-JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00716         h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00717         h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00718     udata[pp + 1] =
00719         h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00720         h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00721         h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00722     udata[pp + 2] =
00723         h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00724         h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00725         h[0] * (((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00726     detC =
00727         -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00728         (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00729         JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00730         JMM[1] * (-JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00731     udata[pp + 0] /= detC;
00732     udata[pp + 1] /= detC;
00733     udata[pp + 2] /= detC;
00734     udata[pp + 3] = h[3];
00735     udata[pp + 4] = h[4];
00736     udata[pp + 5] = h[5];
00737 }
00738 return;
00739 }
```

## 6.30 src/TimeEvolutionFunctions.h File Reference

Functions to propagate data vectors in time according to Maxwell's equations, and various orders in the HE weak-field expansion.

```
#include "LatticePatch.h"
#include "SimulationClass.h"
```

Include dependency graph for TimeEvolutionFunctions.h:



## Data Structures

- class [TimeEvolution](#)  
*monostate TimeEvolution class to propagate the field data in time in a given order of the HE weak-field expansion*

## Functions

- void [linear1DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*Maxwell propagation function for 1D – only for reference.*
- void [nonlinear1DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*HE propagation function for 1D.*
- void [linear2DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*Maxwell propagation function for 2D – only for reference.*
- void [nonlinear2DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*HE propagation function for 2D.*
- void [linear3DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*Maxwell propagation function for 3D – only for reference.*
- void [nonlinear3DProp](#) ([LatticePatch](#) \*data, [N\\_Vector](#) u, [N\\_Vector](#) udot, int \*c)  
*HE propagation function for 3D.*

### 6.30.1 Detailed Description

Functions to propagate data vectors in time according to Maxwell's equations, and various orders in the HE weak-field expansion.

Definition in file [TimeEvolutionFunctions.h](#).

### 6.30.2 Function Documentation

#### 6.30.2.1 linear1DProp()

```
void linear1DProp (
    LatticePatch * data,
    N_Vector u,
    N_Vector udot,
    int * c )
```

Maxwell propagation function for 1D – only for reference.

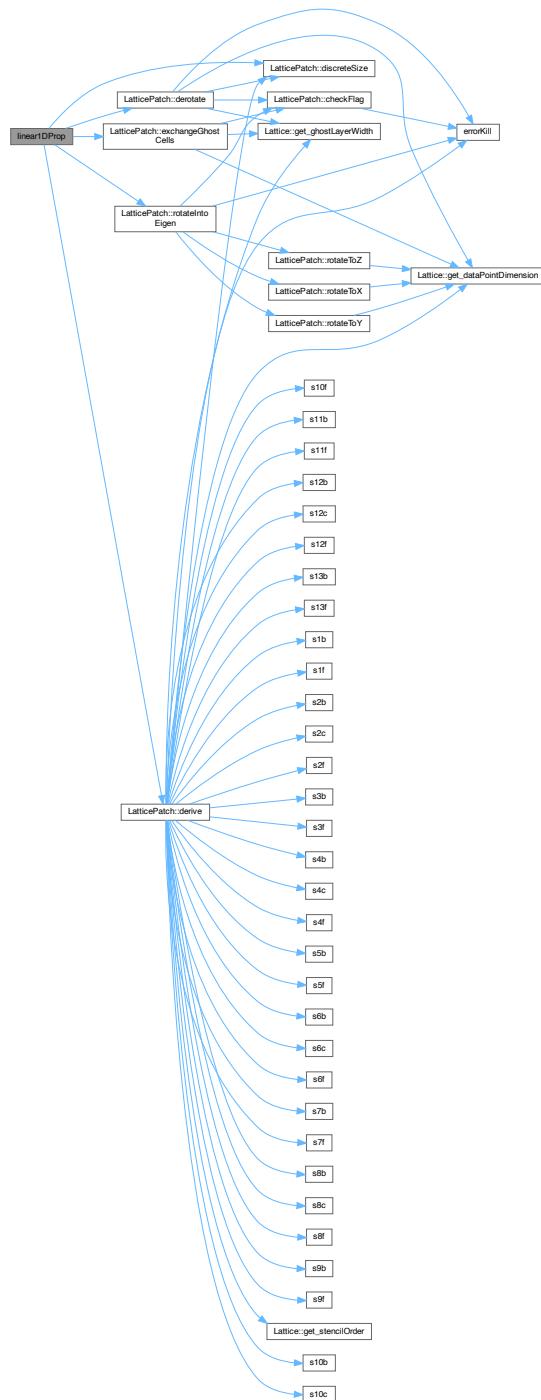
Maxwell propagation function for 1D – only for reference.

Definition at line 48 of file [TimeEvolutionFunctions.cpp](#).

```
00048
00049
00050 // pointers to temporal and spatial derivative data
00051 sunrealtype *duData = data->duData;
00052 sunrealtype *dxData = data->buffData[1 - 1];
00053
00054 // sequence along any dimension according to the scheme:
00055 data->exchangeGhostCells(1); // -> exchange halos
00056 data->rotateIntoEigen(
00057     1); // -> rotate all data to prepare derivative operation
00058 data->derive(1); // -> perform derivative approximation operation on it
00059 data->derotate(
00060     1, dxData); // -> derotate derived data for ensuing time-evolution
00061
00062 const sunindextype totalNP = data->discreteSize();
00063 sunindextype pp = 0;
00064 for (sunindextype i = 0; i < totalNP; i++) {
00065     pp = i * 6;
00066     /*
00067         simple vacuum Maxwell equations for the temporal derivatives using the
00068         spatial derivative only in x-direction without polarization or
00069         magnetization terms
00070     */
00071     duData[pp + 0] = 0;
00072     duData[pp + 1] = -dxData[pp + 5];
00073     duData[pp + 2] = dxData[pp + 4];
00074     duData[pp + 3] = 0;
00075     duData[pp + 4] = dxData[pp + 2];
00076     duData[pp + 5] = -dxData[pp + 1];
00077 }
00078 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [LatticePatch::duData](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Here is the call graph for this function:



### 6.30.2.2 linear2DProp()

```
void linear2DProp (
    LatticePatch * data,
```

```
N_Vector u,  
N_Vector udot,  
int * c )
```

Maxwell propagation function for 2D – only for reference.

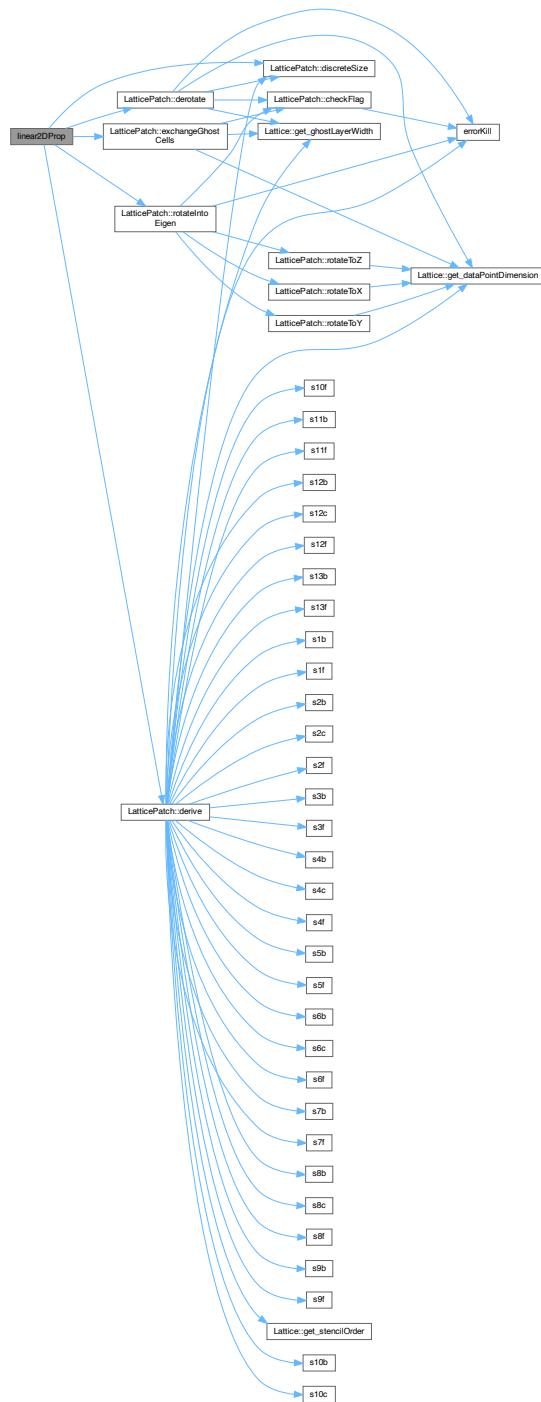
Maxwell propagation function for 2D – only for reference.

Definition at line 296 of file [TimeEvolutionFunctions.cpp](#).

```
00296  
00297  
00298     sunrealtype *duData = data->duData;  
00299     sunrealtype *dxData = data->buffData[1 - 1];  
00300     sunrealtype *dyData = data->buffData[2 - 1];  
00301  
00302     data->exchangeGhostCells(1);  
00303     data->rotateIntoEigen(1);  
00304     data->derive(1);  
00305     data->derotate(1, dxData);  
00306     data->exchangeGhostCells(2);  
00307     data->rotateIntoEigen(2);  
00308     data->derive(2);  
00309     data->derotate(2, dyData);  
00310  
00311     const sunindextype totalNP = data->discreteSize();  
00312     sunindextype pp = 0;  
00313     for (sunindextype i = 0; i < totalNP; i++) {  
00314         pp = i * 6;  
00315         duData[pp + 0] = dyData[pp + 5];  
00316         duData[pp + 1] = -dxData[pp + 5];  
00317         duData[pp + 2] = -dyData[pp + 3] + dxData[pp + 4];  
00318         duData[pp + 3] = -dyData[pp + 2];  
00319         duData[pp + 4] = dxData[pp + 2];  
00320         duData[pp + 5] = dyData[pp + 0] - dxData[pp + 1];  
00321     }  
00322 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [LatticePatch::duData](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Here is the call graph for this function:



### 6.30.2.3 linear3DProp()

```
void linear3DProp (
    LatticePatch * data,
```

```
N_Vector u,  
N_Vector udot,  
int * c )
```

Maxwell propagation function for 3D – only for reference.

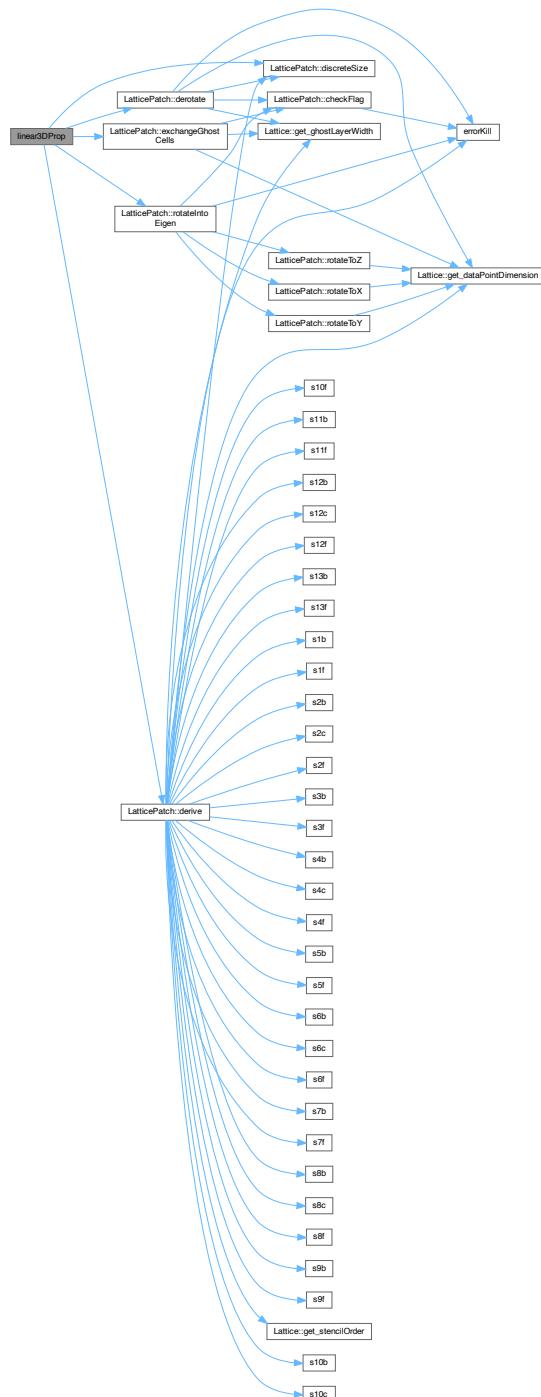
Maxwell propagation function for 3D – only for reference.

Definition at line 509 of file [TimeEvolutionFunctions.cpp](#).

```
00509  
00510  
00511 sunrealtyp *duData = data->duData;  
00512 sunrealtyp *dxData = data->buffData[1 - 1];  
00513 sunrealtyp *dyData = data->buffData[2 - 1];  
00514 sunrealtyp *dzData = data->buffData[3 - 1];  
00515  
00516 data->exchangeGhostCells(1);  
00517 data->rotateIntoEigen(1);  
00518 data->derive(1);  
00519 data->derotate(1, dxData);  
00520 data->exchangeGhostCells(2);  
00521 data->rotateIntoEigen(2);  
00522 data->derive(2);  
00523 data->derotate(2, dyData);  
00524 data->exchangeGhostCells(3);  
00525 data->rotateIntoEigen(3);  
00526 data->derive(3);  
00527 data->derotate(3, dzData);  
00528  
00529 const sunindextype totalNP = data->discreteSize();  
00530 sunindextype pp = 0;  
00531 for (sunindextype i = 0; i < totalNP; i++) {  
00532     pp = i * 6;  
00533     duData[pp + 0] = dyData[pp + 5] - dzData[pp + 4];  
00534     duData[pp + 1] = dzData[pp + 3] - dxData[pp + 5];  
00535     duData[pp + 2] = dxData[pp + 4] - dyData[pp + 3];  
00536     duData[pp + 3] = -dyData[pp + 2] + dzData[pp + 1];  
00537     duData[pp + 4] = -dzData[pp + 0] + dxData[pp + 2];  
00538     duData[pp + 5] = -dxData[pp + 1] + dyData[pp + 0];  
00539 }  
00540 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [LatticePatch::duData](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Here is the call graph for this function:



#### 6.30.2.4 nonlinear1DProp()

```
void nonlinear1DProp (
    LatticePatch * data,
```

```
N_Vector u,
N_Vector udot,
int * c )
```

HE propagation function for 1D.

HE propagation function for 1D.

Definition at line 81 of file TimeEvolutionFunctions.cpp.

```
00081
00082
00083 // NVector pointers to provided field values and their temp. derivatives
00084 #if defined(_OPENMP)
00085 sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00086             *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00087 #else
00088 sunrealtype *udata = NV_DATA_P(u),
00089             *dudata = NV_DATA_P(udot);
00090 #endif
00091
00092 // pointer to spatial derivatives via pach data
00093 sunrealtype *dxData = data->buffData[1 - 1];
00094
00095 // same sequence as in the linear case
00096 data->exchangeGhostCells(1);
00097 data->rotateIntoEigen(1);
00098 data->derive(1);
00099 data->derotate(1, dxData);
00100
00101 /*
00102 F and G are nonzero in the nonlinear case,
00103 polarization and magnetization derivatives
00104 w.r.t. E- and B-field go into the e.o.m.
00105 */
00106 static sunrealtype f, g; // em field invariants F, G
00107 // derivatives of HE Lagrangian w.r.t. field invariants
00108 static sunrealtype lf, lff, lfg, lg, lgg;
00109 // matrix to hold derivatives of polarization and magnetization
00110 static std::array<sunrealtype, 21> JMM;
00111 // array to hold E^2 and B^2 components
00112 static std::array<sunrealtype, 6> Quad;
00113 // array to hold intermediate temp. derivatives of E and B
00114 static std::array<sunrealtype, 6> h;
00115 // determinant needed for explicit matrix inversion
00116 static sunrealtype detC = nan("0x12345");
00117
00118 // number of points in the patch
00119 const sunindextype totalNP = data->discreteSize();
00120 #pragma omp parallel for default(none) \
00121 private(JMM, Quad, h, detC) \
00122 shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, dxData) \
00123 schedule(static)
00124 for (sunindextype pp = 0; pp < totalNP * 6;
00125     pp += 6) { // loop over all 6dim points in the patch
00126     // em field Lorentz invariants F and G
00127     f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00128                 (Quad[1] = udata[pp + 1] * udata[pp + 1]) +
00129                 (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00130                 (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00131                 (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00132                 (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00133     g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00134         udata[pp + 2] * udata[pp + 5];
00135     // process/expansion order and corresponding derivative values of L
00136     // w.r.t. F, G
00137     switch (*c) {
00138     case 0: // linear Maxwell vacuum
00139         lf = 0;
00140         lff = 0;
00141         lfg = 0;
00142         lg = 0;
00143         lgg = 0;
00144         break;
00145     case 1: // only 4-photon processes
00146         lf = 0.000206527095658582755255648 * f;
00147         lff = 0.000206527095658582755255648;
00148         lfg = 0;
00149         lg = 0.0003614224174025198216973841 * g;
00150         lgg = 0.0003614224174025198216973841;
00151         break;
00152     case 2: // only 6-photon processes
00153         lf = 0.000354046449700427580438254 * f * f +
00154             0.000191775160254398272737387 * g * g;
```

```

00155     lff = 0.0007080928994008551608765075 * f;
00156     lfg = 0.0003835503205087965454747749 * g;
00157     lg = 0.0003835503205087965454747749 * f * g;
00158     lgg = 0.0003835503205087965454747749 * f;
00159     break;
00160 case 3: // 4- and 6-photon processes
00161     lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00162         f +
00163         0.000191775160254398272737387 * g * g;
00164     lff = 0.000206527095658582755255648 + 0.0007080928994008551608765075 * f;
00165     lfg = 0.0003835503205087965454747749 * g;
00166     lg = (0.0003614224174025198216973841 +
00167         0.0003835503205087965454747749 * f) *
00168         g;
00169     lgg = 0.0003614224174025198216973841 + 0.0003835503205087965454747749 * f;
00170     break;
00171 default:
00172     errorKill(
00173         "You need to specify a correct order in the weak-field expansion.");
00174 }
00175
00176 // derivatives of polarization and magnetization w.r.t. E and B
00177 // Jpx(Ex)
00178 JMM[0] = lf + lff * Quad[0] +
00179     udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00180 // Jpx(Ey)
00181 JMM[1] =
00182     lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00183     lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00184 // Jpy(Ey)
00185 JMM[2] = lf + lff * Quad[1] +
00186     udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00187 // Jpx(Ez) = Jpz(Ex)
00188 JMM[3] =
00189     lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00190     lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00191 // Jpy(Ez) = Jpz(Ey)
00192 JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00193     lfg * udata[2 + pp] * udata[4 + pp] +
00194     lfg * udata[1 + pp] * udata[5 + pp] +
00195     lgg * udata[4 + pp] * udata[5 + pp];
00196 // Jpz(Ez)
00197 JMM[5] = lf + lff * Quad[2] +
00198     udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00199 // Jpx(Bx) = Jmx(Ex)
00200 JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00201     (-lff + lgg) * udata[pp] * udata[3 + pp];
00202 // Jpy(Bx) = Jmx(Ey)
00203 JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00204     udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00205 // Jpz(Bx) = Jmx(Ez)
00206 JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00207     udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00208 // Jmx(Bx)
00209 JMM[9] = -lf + lgg * Quad[0] +
00210     udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00211 // Jpx(By) = Jmy(Ex)
00212 JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00213     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00214 // Jpy(By) = Jmy(Ey)
00215 JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00216     (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00217 // Jpz(By) = Jmy(Ez)
00218 JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00219     udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00220 // Jmx(By) = Jmy(Bx)
00221 JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00222     lff * udata[3 + pp] * udata[4 + pp] -
00223     lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00224 // Jmy(By)
00225 JMM[14] = -lf + lgg * Quad[1] +
00226     udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00227 // Jmz(Ex) = Jpx(Bz)
00228 JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00229     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00230 // Jmz(Ey) = Jpy(Bz)
00231 JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00232     (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00233 // Jpz(Bz) = Jmx(Ez)
00234 JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00235     (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00236 // Jmz(Bx) = Jmx(Bz)
00237 JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00238     lff * udata[3 + pp] * udata[5 + pp] -
00239     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00240 // Jmy(Bz) = Jmz(By)
00241 JMM[19] =

```

```

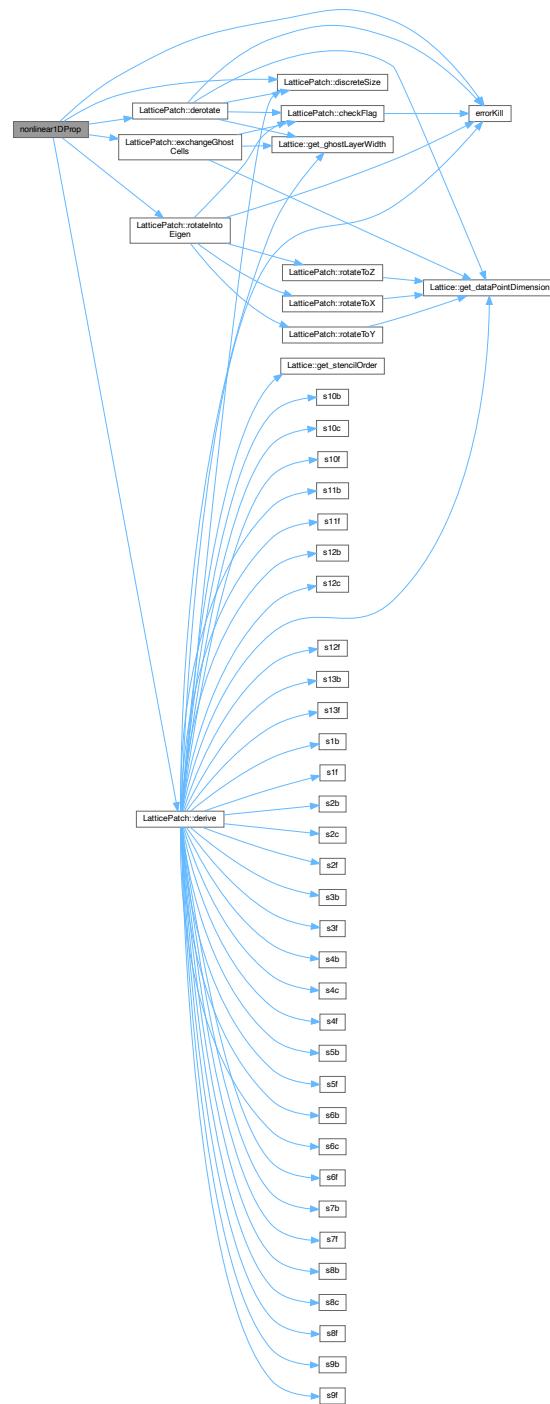
00242     lgg * udata[1 + pp] * udata[2 + pp] +
00243     lff * udata[4 + pp] * udata[5 + pp] -
00244     lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]));
00245 // Jmz(Bz)
00246 JMM[20] = -lf + lgg * Quad[2] +
00247     udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00248
00249 // apply Z
00250 // top block: -QJm(E)*E, Q-QJm(B)*B
00251 h[0] = 0;
00252 h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00253     dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00254     dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00255 h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00256     dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00257     dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00258 // bottom blocks: -Q*E
00259 h[3] = 0;
00260 h[4] = dxData[2 + pp];
00261 h[5] = -dxData[1 + pp];
00262 // (1+A)^-1 applies only to E components
00263 // -Jp(B)*B
00264 h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00265 h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00266 h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00267 // apply C^-1 explicitly, with C=1+Jp(E)
00268 dudata[pp + 0] =
00269     h[2] * (- (JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00270     h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00271     h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00272 dudata[pp + 1] =
00273     h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00274     h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00275     h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00276 dudata[pp + 2] =
00277     h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00278     h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00279     h[0] * (((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00280 detC = // determinant of C
00281     -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00282     (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00283     JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00284     JMM[1] * (- (JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00285 dudata[pp + 0] /= detC;
00286 dudata[pp + 1] /= detC;
00287 dudata[pp + 2] /= detC;
00288 dudata[pp + 3] = h[3];
00289 dudata[pp + 4] = h[4];
00290 dudata[pp + 5] = h[5];
00291 }
00292 return;
00293 }

```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Referenced by [Sim1D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.30.2.5 nonlinear2DProp()

```
void nonlinear2DProp (
    LatticePatch * data,
    N_Vector u,
    N_Vector udot,
    int * c )
```

HE propagation function for 2D.

HE propagation function for 2D.

Definition at line 325 of file [TimeEvolutionFunctions.cpp](#).

```
00325
00326
00327 #if defined(_OPENMP)
00328     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00329                 *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00330 #else
00331     sunrealtype *udata = NV_DATA_P(u),
00332                 *dudata = NV_DATA_P(udot);
00333 #endif
00334
00335     sunrealtype *dxData = data->buffData[1 - 1];
00336     sunrealtype *dyData = data->buffData[2 - 1];
00337
00338     data->exchangeGhostCells(1);
00339     data->rotateIntoEigen(1);
00340     data->derive(1);
00341     data->derotate(1, dxData);
00342     data->exchangeGhostCells(2);
00343     data->rotateIntoEigen(2);
00344     data->derive(2);
00345     data->derotate(2, dyData);
00346
00347     static sunrealtype f, g;
00348     static sunrealtype lf, lff, lfg, lg, lgg;
00349     static std::array<sunrealtype, 21> JMM;
00350     static std::array<sunrealtype, 6> Quad;
00351     static std::array<sunrealtype, 6> h;
00352     static sunrealtype detC;
00353
00354     const sunindextype totalNP = data->discreteSize();
00355     #pragma omp parallel for default(none) \
00356     private(JMM, Quad, h, detC) \
00357     shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, \
00358             dxData, dyData) \
00359     schedule(static)
00360     for (sunindextype pp = 0; pp < totalNP * 6; pp += 6) {
00361         f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00362                     (Quad[1] = udata[pp + 1] * udata[pp + 1])) +
00363                     (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00364                     (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00365                     (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00366                     (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00367         g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
```

```

00368     udata[pp + 2] * udata[pp + 5];
00369     switch (*c) {
00370     case 0:
00371         lff = 0;
00372         lff = 0;
00373         lfg = 0;
00374         lg = 0;
00375         lgg = 0;
00376         break;
00377     case 1:
00378         lf = 0.000206527095658582755255648 * f;
00379         lff = 0.000206527095658582755255648;
00380         lfg = 0;
00381         lg = 0.0003614224174025198216973841 * g;
00382         lgg = 0.0003614224174025198216973841;
00383         break;
00384     case 2:
00385         lf = 0.000354046449700427580438254 * f * f +
00386             0.000191775160254398272737387 * g * g;
00387         lff = 0.0007080928994008551608765075 * f;
00388         lfg = 0.0003835503205087965454747749 * g;
00389         lg = 0.0003835503205087965454747749 * f * g;
00390         lgg = 0.0003835503205087965454747749 * f;
00391         break;
00392     case 3:
00393         lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00394             f +
00395             0.000191775160254398272737387 * g * g;
00396         lff = 0.000206527095658582755255648 + 0.000708092899400855160876508 * f;
00397         lfg = 0.0003835503205087965454747749 * g;
00398         lg = (0.000361422417402519821697384 + 0.000383550320508796545474775 * f) *
00399             g;
00400         lgg = 0.000361422417402519821697384 + 0.000383550320508796545474775 * f;
00401         break;
00402     default:
00403         errorKill(
00404             "You need to specify a correct order in the weak-field expansion.");
00405     }
00406
00407 JMM[0] = lf + lff * Quad[0] +
00408     udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00409 JMM[1] =
00410     lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00411     lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00412 JMM[2] = lf + lff * Quad[1] +
00413     udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00414 JMM[3] =
00415     lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00416     lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00417 JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00418     lfg * udata[2 + pp] * udata[4 + pp] +
00419     lfg * udata[1 + pp] * udata[5 + pp] +
00420     lgg * udata[4 + pp] * udata[5 + pp];
00421 JMM[5] = lf + lff * Quad[2] +
00422     udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00423 JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00424     (-lff + lgg) * udata[pp] * udata[3 + pp];
00425 JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00426     udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00427 JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00428     udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00429 JMM[9] = -lf + lgg * Quad[0] +
00430     udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00431 JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00432     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00433 JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00434     (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00435 JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00436     udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00437 JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00438     lff * udata[3 + pp] * udata[4 + pp] -
00439     lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00440 JMM[14] = -lf + lgg * Quad[1] +
00441     udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00442 JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00443     (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00444 JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00445     (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00446 JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00447     (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00448 JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00449     lff * udata[3 + pp] * udata[5 + pp] -
00450     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00451 JMM[19] =
00452     lgg * udata[1 + pp] * udata[2 + pp] +
00453     lff * udata[4 + pp] * udata[5 + pp] -
00454     lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);

```

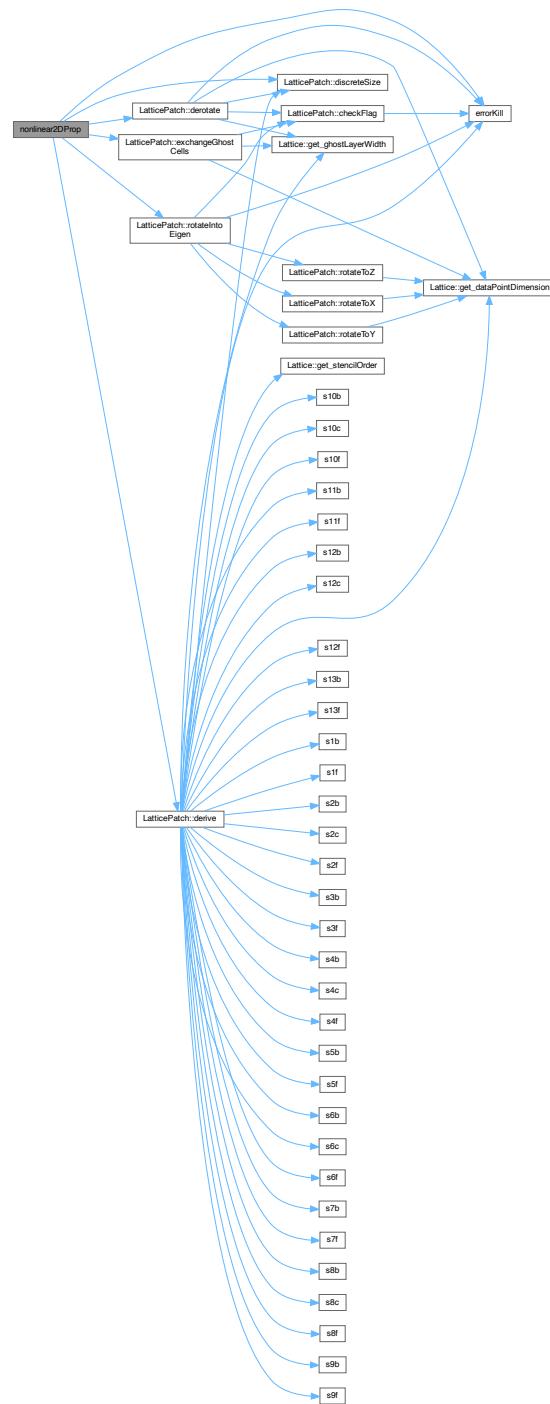
```

00455     JMM[20] = -lf + lgg * Quad[2] +
00456             udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00457
00458     h[0] = 0;
00459     h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00460             dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00461             dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00462     h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00463             dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00464             dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00465     h[3] = 0;
00466     h[4] = dxData[2 + pp];
00467     h[5] = -dxData[1 + pp];
00468     h[0] += -(dyData[pp] * JMM[15]) - dyData[1 + pp] * JMM[16] -
00469             dyData[2 + pp] * JMM[17] - dyData[3 + pp] * JMM[18] -
00470             dyData[4 + pp] * JMM[19] + dyData[5 + pp] * (1 - JMM[20]);
00471     h[1] += 0;
00472     h[2] += dyData[pp] * JMM[6] + dyData[1 + pp] * JMM[7] +
00473             dyData[2 + pp] * JMM[8] + dyData[3 + pp] * (-1 + JMM[9]) +
00474             dyData[4 + pp] * JMM[13] + dyData[5 + pp] * JMM[18];
00475     h[3] += -dyData[2 + pp];
00476     h[4] += 0;
00477     h[5] += dyData[pp];
00478     h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00479     h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00480     h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00481     dudata[pp + 0] =
00482         h[2] * (- (JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00483         h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00484         h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00485     dudata[pp + 1] =
00486         h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00487         h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00488         h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00489     dudata[pp + 2] =
00490         h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00491         h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00492         h[0] * (-((1 + JMM[2]) * JMM[3]) + JMM[1] * JMM[4]);
00493     detC =
00494         -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00495         (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00496         JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00497         JMM[1] * (- (JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00498     dudata[pp + 0] /= detC;
00499     dudata[pp + 1] /= detC;
00500     dudata[pp + 2] /= detC;
00501     dudata[pp + 3] = h[3];
00502     dudata[pp + 4] = h[4];
00503     dudata[pp + 5] = h[5];
00504 }
00505 return;
00506 }
```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Referenced by [Sim2D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.30.2.6 nonlinear3DProp()

```
void nonlinear3DProp (  
    LatticePatch * data,  
    N_Vector u,  
    N_Vector udot,  
    int * c )
```

## HE propagation function for 3D.

## HE propagation function for 3D.

Definition at line 543 of file [TimeEvolutionFunctions.cpp](#).

```

00543
00544
00545 #if defined(_OPENMP)
00546     sunrealtype *udata = N_VGetArrayPointer_MPIPlusX(u),
00547             *dudata = N_VGetArrayPointer_MPIPlusX(udot);
00548 #else
00549     sunrealtype *udata = NV_DATA_P(u),
00550             *dudata = NV_DATA_P(udot);
00551 #endif
00552
00553     sunrealtype *dxData = data->buffData[1 - 1];
00554     sunrealtype *dyData = data->buffData[2 - 1];
00555     sunrealtype *dzData = data->buffData[3 - 1];
00556
00557     data->exchangeGhostCells(1);
00558     data->rotateIntoEigen(1);
00559     data->derive(1);
00560     data->derotate(1,dxData);
00561     data->exchangeGhostCells(2);
00562     data->rotateIntoEigen(2);
00563     data->derive(2);
00564     data->derotate(2,dyData);
00565     data->exchangeGhostCells(3);
00566     data->rotateIntoEigen(3);
00567     data->derive(3);
00568     data->derotate(3,dzData);
00569
00570     static sunrealtype f, g;
00571     static sunrealtype lf, lff, lfg, lg, lgg;
00572     static std::array<sunrealtype, 2> JMM;
00573     static std::array<sunrealtype, 6> Quad;
00574     static std::array<sunrealtype, 6> h;
00575     static sunrealtype detC = nan("0x12345");
00576
00577     const sunindextype totalNP = data->discreteSize();
00578 #pragma omp parallel for default(none) \
00579 private(JMM, Quad, h, detC) \
00580 shared(totalNP, c, f, g, lf, lff, lfg, lg, lgg, udata, dudata, \
00581         dxData, dyData, dzData) \
00582 schedule(static)
00583 for (sunindextype pp = 0; pp < totalNP * 6; pp += 6) {
00584     f = 0.5 * ((Quad[0] = udata[pp] * udata[pp]) +
00585                 (Quad[1] = udata[pp + 1] * udata[pp + 1]) +

```

```

00586             (Quad[2] = udata[pp + 2] * udata[pp + 2]) -
00587             (Quad[3] = udata[pp + 3] * udata[pp + 3]) -
00588             (Quad[4] = udata[pp + 4] * udata[pp + 4]) -
00589             (Quad[5] = udata[pp + 5] * udata[pp + 5]));
00590     g = udata[pp] * udata[pp + 3] + udata[pp + 1] * udata[pp + 4] +
00591         udata[pp + 2] * udata[pp + 5];
00592     switch (*c) {
00593     case 0:
00594         lf = 0;
00595         lff = 0;
00596         lfg = 0;
00597         lg = 0;
00598         lgg = 0;
00599         break;
00600     case 1:
00601         lf = 0.000206527095658582755255648 * f;
00602         lff = 0.000206527095658582755255648;
00603         lfg = 0;
00604         lg = 0.0003614224174025198216973841 * g;
00605         lgg = 0.0003614224174025198216973841;
00606         break;
00607     case 2:
00608         lf = 0.000354046449700427580438254 * f * f +
00609             0.000191775160254398272737387 * g * g;
00610         lff = 0.0007080928994008551608765075 * f;
00611         lfg = 0.0003835503205087965454747749 * g;
00612         lg = 0.0003835503205087965454747749 * f * g;
00613         lgg = 0.0003835503205087965454747749 * f;
00614         break;
00615     case 3:
00616         lf = (0.000206527095658582755255648 + 0.000354046449700427580438254 * f) *
00617             f +
00618             0.000191775160254398272737387 * g * g;
00619         lff = 0.000206527095658582755255648 + 0.000708092899400855160876508 * f;
00620         lfg = 0.0003835503205087965454747749 * g;
00621         lg = (0.000361422417402519821697384 + 0.000383550320508796545474775 * f) *
00622             g;
00623         lgg = 0.000361422417402519821697384 + 0.000383550320508796545474775 * f;
00624         break;
00625     default:
00626         errorKill(
00627             "You need to specify a correct order in the weak-field expansion.");
00628     }
00629
00630     JMM[0] = lf + lff * Quad[0] +
00631         udata[3 + pp] * (2 * lfg * udata[pp] + lgg * udata[3 + pp]);
00632     JMM[1] =
00633         lff * udata[pp] * udata[1 + pp] + lfg * udata[1 + pp] * udata[3 + pp] +
00634         lfg * udata[pp] * udata[4 + pp] + lgg * udata[3 + pp] * udata[4 + pp];
00635     JMM[2] = lf + lff * Quad[1] +
00636         udata[4 + pp] * (2 * lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00637     JMM[3] =
00638         lff * udata[pp] * udata[2 + pp] + lfg * udata[2 + pp] * udata[3 + pp] +
00639         lfg * udata[pp] * udata[5 + pp] + lgg * udata[3 + pp] * udata[5 + pp];
00640     JMM[4] = lff * udata[1 + pp] * udata[2 + pp] +
00641             lfg * udata[2 + pp] * udata[4 + pp] +
00642             lfg * udata[1 + pp] * udata[5 + pp] +
00643             lgg * udata[4 + pp] * udata[5 + pp];
00644     JMM[5] = lf + lff * Quad[2] +
00645         udata[5 + pp] * (2 * lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00646     JMM[6] = lg + lfg * (Quad[0] - Quad[3 + 0]) +
00647             (-lff + lgg) * udata[pp] * udata[3 + pp];
00648     JMM[7] = -(udata[3 + pp] * (lff * udata[1 + pp] + lfg * udata[4 + pp])) +
00649             udata[pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]);
00650     JMM[8] = -(udata[3 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00651             udata[pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00652     JMM[9] = -lf + lgg * Quad[0] +
00653         udata[3 + pp] * (-2 * lfg * udata[pp] + lff * udata[3 + pp]);
00654     JMM[10] = udata[1 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00655             (lff * udata[pp] + lfg * udata[3 + pp]) * udata[4 + pp];
00656     JMM[11] = lg + lfg * (Quad[1] - Quad[4 + 0]) +
00657             (-lff + lgg) * udata[1 + pp] * udata[4 + pp];
00658     JMM[12] = -(udata[4 + pp] * (lff * udata[2 + pp] + lfg * udata[5 + pp])) +
00659             udata[1 + pp] * (lfg * udata[2 + pp] + lgg * udata[5 + pp]);
00660     JMM[13] = lgg * udata[pp] * udata[1 + pp] +
00661         lff * udata[3 + pp] * udata[4 + pp] -
00662         lfg * (udata[1 + pp] * udata[3 + pp] + udata[pp] * udata[4 + pp]);
00663     JMM[14] = -lf + lgg * Quad[1] +
00664         udata[4 + pp] * (-2 * lfg * udata[1 + pp] + lff * udata[4 + pp]);
00665     JMM[15] = udata[2 + pp] * (lfg * udata[pp] + lgg * udata[3 + pp]) -
00666             (lff * udata[pp] + lfg * udata[3 + pp]) * udata[5 + pp];
00667     JMM[16] = udata[2 + pp] * (lfg * udata[1 + pp] + lgg * udata[4 + pp]) -
00668             (lff * udata[1 + pp] + lfg * udata[4 + pp]) * udata[5 + pp];
00669     JMM[17] = lg + lfg * (Quad[2] - Quad[5 + 0]) +
00670             (-lff + lgg) * udata[2 + pp] * udata[5 + pp];
00671     JMM[18] = lgg * udata[pp] * udata[2 + pp] +
00672             lff * udata[3 + pp] * udata[5 + pp] -

```

```

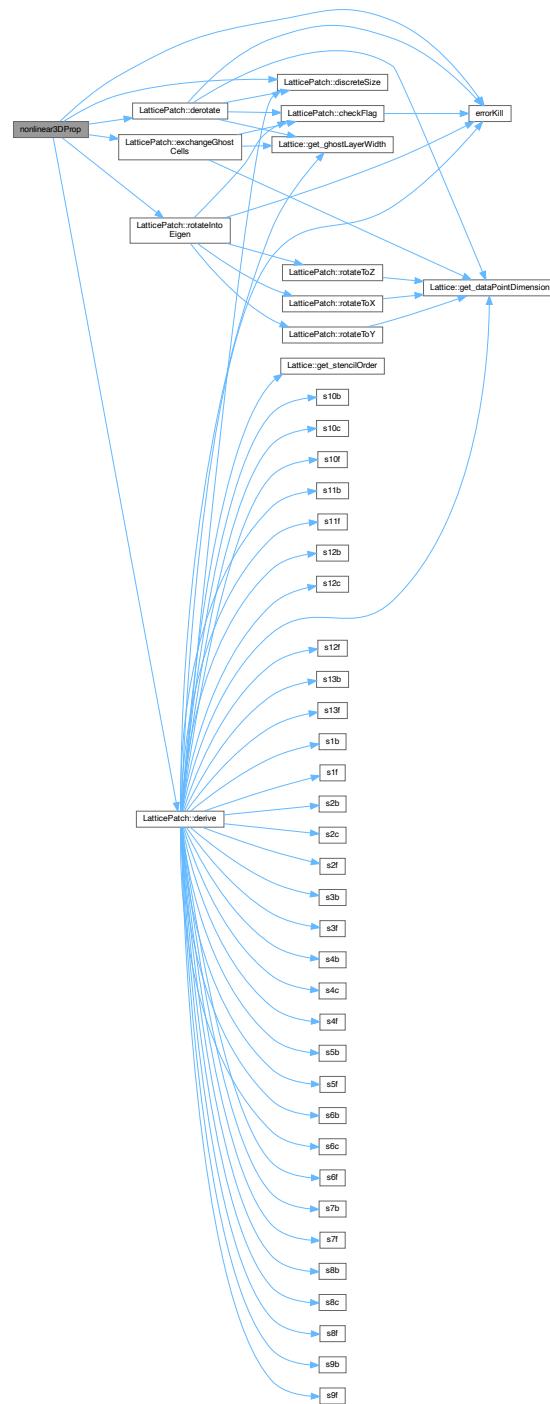
00673     lfg * (udata[2 + pp] * udata[3 + pp] + udata[pp] * udata[5 + pp]);
00674     JMM[19] =
00675         lgg * udata[1 + pp] * udata[2 + pp] +
00676         lff * udata[4 + pp] * udata[5 + pp] -
00677         lfg * (udata[2 + pp] * udata[4 + pp] + udata[1 + pp] * udata[5 + pp]);
00678     JMM[20] = -lf + lgg * Quad[2] +
00679         udata[5 + pp] * (-2 * lfg * udata[2 + pp] + lff * udata[5 + pp]);
00680
00681     h[0] = 0;
00682     h[1] = dxData[pp] * JMM[15] + dxData[1 + pp] * JMM[16] +
00683         dxData[2 + pp] * JMM[17] + dxData[3 + pp] * JMM[18] +
00684         dxData[4 + pp] * JMM[19] + dxData[5 + pp] * (-1 + JMM[20]);
00685     h[2] = -(dxData[pp] * JMM[10]) - dxData[1 + pp] * JMM[11] -
00686         dxData[2 + pp] * JMM[12] - dxData[3 + pp] * JMM[13] +
00687         dxData[4 + pp] * (1 - JMM[14]) - dxData[5 + pp] * JMM[19];
00688     h[3] = 0;
00689     h[4] = dxData[2 + pp];
00690     h[5] = -dxData[1 + pp];
00691     h[0] += -(dyData[pp] * JMM[15]) - dyData[1 + pp] * JMM[16] -
00692         dyData[2 + pp] * JMM[17] - dyData[3 + pp] * JMM[18] -
00693         dyData[4 + pp] * JMM[19] + dyData[5 + pp] * (1 - JMM[20]);
00694     h[1] += 0;
00695     h[2] += dyData[pp] * JMM[6] + dyData[1 + pp] * JMM[7] +
00696         dyData[2 + pp] * JMM[8] + dyData[3 + pp] * (-1 + JMM[9]) +
00697         dyData[4 + pp] * JMM[13] + dyData[5 + pp] * JMM[18];
00698     h[3] += -dyData[2 + pp];
00699     h[4] += 0;
00700     h[5] += dyData[pp];
00701     h[0] += dzData[pp] * JMM[10] + dzData[1 + pp] * JMM[11] +
00702         dzData[2 + pp] * JMM[12] + dzData[3 + pp] * JMM[13] +
00703         dzData[4 + pp] * (-1 + JMM[14]) + dzData[5 + pp] * JMM[19];
00704     h[1] += -(dzData[pp] * JMM[6]) - dzData[1 + pp] * JMM[7] -
00705         dzData[2 + pp] * JMM[8] + dzData[3 + pp] * (1 - JMM[9]) -
00706         dzData[4 + pp] * JMM[13] - dzData[5 + pp] * JMM[18];
00707     h[2] += 0;
00708     h[3] += dzData[1 + pp];
00709     h[4] += -dzData[pp];
00710     h[5] += 0;
00711     h[0] -= h[3] * JMM[6] + h[4] * JMM[10] + h[5] * JMM[15];
00712     h[1] -= h[3] * JMM[7] + h[4] * JMM[11] + h[5] * JMM[16];
00713     h[2] -= h[3] * JMM[8] + h[4] * JMM[12] + h[5] * JMM[17];
00714     dudata[pp + 0] =
00715         h[2] * (-JMM[3] * (1 + JMM[2])) + JMM[1] * JMM[4]) +
00716         h[1] * (JMM[3] * JMM[4] - JMM[1] * (1 + JMM[5])) +
00717         h[0] * (1 - JMM[4] * JMM[4] + JMM[5] + JMM[2] * (1 + JMM[5]));
00718     dudata[pp + 1] =
00719         h[2] * (JMM[3] * JMM[1] - (1 + JMM[0]) * JMM[4]) +
00720         h[1] * (1 - JMM[3] * JMM[3] + JMM[5] + JMM[0] * (1 + JMM[5])) +
00721         h[0] * (JMM[4] * JMM[3] - JMM[1] * (1 + JMM[5]));
00722     dudata[pp + 2] =
00723         h[2] * (1 - JMM[1] * JMM[1] + JMM[2] + JMM[0] * (1 + JMM[2])) +
00724         h[1] * (JMM[1] * JMM[3] - (1 + JMM[0]) * JMM[4]) +
00725         h[0] * (-(1 + JMM[2]) * JMM[3] + JMM[1] * JMM[4]);
00726     detC =
00727         -((1 + JMM[2]) * (-1 + JMM[3] * JMM[3])) +
00728         (JMM[3] * JMM[1] - JMM[4]) * JMM[4] + JMM[5] + JMM[2] * JMM[5] +
00729         JMM[0] * (1 + JMM[2] - JMM[4] * JMM[4] + (1 + JMM[2]) * JMM[5]) -
00730         JMM[1] * (-(JMM[4] * JMM[3]) + JMM[1] * (1 + JMM[5]));
00731     dudata[pp + 0] /= detC;
00732     dudata[pp + 1] /= detC;
00733     dudata[pp + 2] /= detC;
00734     dudata[pp + 3] = h[3];
00735     dudata[pp + 4] = h[4];
00736     dudata[pp + 5] = h[5];
00737 }
00738 return;
00739 }

```

References [LatticePatch::buffData](#), [LatticePatch::derive\(\)](#), [LatticePatch::derotate\(\)](#), [LatticePatch::discreteSize\(\)](#), [errorKill\(\)](#), [LatticePatch::exchangeGhostCells\(\)](#), and [LatticePatch::rotateIntoEigen\(\)](#).

Referenced by [Sim3D\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.31 TimeEvolutionFunctions.h

[Go to the documentation of this file.](#)

```

00001 ///////////////////////////////////////////////////////////////////
00002 /// @file TimeEvolutionFunctions.h
00003 /// @brief Functions to propagate data vectors in time
00004 /// according to Maxwell's equations, and various
00005 /// orders in the HE weak-field expansion
00006 ///////////////////////////////////////////////////////////////////
00007
00008 #pragma once
00009
00010 #include "LatticePatch.h"
00011 #include "SimulationClass.h"
00012
00013 /** @brief monostate TimeEvolution class to propagate the field data in time in
00014 * a given order of the HE weak-field expansion */
00015 class TimeEvolution {
00016 public:
00017     /// choice which processes of the weak field expansion are included
00018     static int *c;
00019
00020     /// Pointer to functions for differentiation and time evolution
00021     static void (*TimeEvolver)(LatticePatch *, N_Vector, N_Vector, int *);
00022
00023     /// CVODE right hand side function (CVRhsFn) to provide IVP of the ODE
00024     static int f(sunrealtype t, N_Vector u, N_Vector udot, void *data_loc);
00025 };
00026
00027 /// Maxwell propagation function for 1D -- only for reference
00028 void linear1DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c);
00029 /// HE propagation function for 1D
00030 void nonlinear1DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c);
00031 /// Maxwell propagation function for 2D -- only for reference
00032 void linear2DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c);
00033 /// HE propagation function for 2D
00034 void nonlinear2DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c);
00035 /// Maxwell propagation function for 3D -- only for reference
00036 void linear3DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c);
00037 /// HE propagation function for 3D
00038 void nonlinear3DProp(LatticePatch *data, N_Vector u, N_Vector udot, int *c);
00039
  
```



# Index

~LatticePatch  
    LatticePatch, 61

~Simulation  
    Simulation, 118

A1  
    Gauss2D, 20  
    Gauss3D, 25

A2  
    Gauss2D, 20  
    Gauss3D, 26

add  
    ICSetter, 35

addGauss1D  
    ICSetter, 36

addGauss2D  
    ICSetter, 37

addGauss3D  
    ICSetter, 37

addInitialConditions  
    Simulation, 119

addPeriodicCLayerInX  
    Simulation, 120

addPeriodicCLayerInXY  
    Simulation, 120

addPlaneWave1D  
    ICSetter, 38

addPlaneWave2D  
    ICSetter, 39

addPlaneWave3D  
    ICSetter, 39

addToSpace  
    Gauss1D, 13  
    Gauss2D, 19  
    Gauss3D, 25  
    PlaneWave1D, 109  
    PlaneWave2D, 112  
    PlaneWave3D, 115

advanceToTime  
    Simulation, 121

Amp  
    Gauss2D, 20  
    Gauss3D, 26

amp  
    gaussian2D, 31  
    gaussian3D, 33

axis  
    Gauss2D, 20  
    Gauss3D, 26  
    gaussian2D, 31

    gaussian3D, 33

buffData  
    LatticePatch, 84

BuffersInitialized  
    LatticePatch.h, 207

buffX  
    LatticePatch, 84

buffY  
    LatticePatch, 84

buffZ  
    LatticePatch, 84

c  
    TimeEvolution, 137

check\_error  
    LatticePatch.cpp, 188  
    LatticePatch.h, 205

check\_retval  
    LatticePatch.cpp, 188  
    LatticePatch.h, 205

checkFlag  
    LatticePatch, 61  
    Simulation, 122

checkNoFlag  
    Simulation, 124

comm  
    Lattice, 53

cvode\_mem  
    Simulation, 133

CvodeObjectSetUp  
    SimulationClass.h, 233

dataPointDimension  
    Lattice, 53

DerivationStencils.h  
    s10b, 142  
    s10c, 143  
    s10f, 144  
    s11b, 145  
    s11f, 146  
    s12b, 147  
    s12c, 148  
    s12f, 149  
    s13b, 150  
    s13f, 151  
    s1b, 152  
    s1f, 153  
    s2b, 154  
    s2c, 155

s2f, 156  
 s3b, 157  
 s3f, 158  
 s4b, 159  
 s4c, 160  
 s4f, 161  
 s5b, 162  
 s5f, 163  
 s6b, 164  
 s6c, 165  
 s6f, 166  
 s7b, 167  
 s7f, 168  
 s8b, 169  
 s8c, 170  
 s8f, 171  
 s9b, 172  
 s9f, 173  
 derive  
     LatticePatch, 63  
 derotate  
     LatticePatch, 68  
 dis  
     Gauss2D, 21  
     Gauss3D, 26  
 discreteSize  
     LatticePatch, 70  
 du  
     LatticePatch, 84  
 duData  
     LatticePatch, 85  
 duLocal  
     LatticePatch, 85  
 dx  
     Lattice, 53  
     LatticePatch, 85  
 dy  
     Lattice, 53  
     LatticePatch, 85  
 dz  
     Lattice, 53  
     LatticePatch, 86  
 envelopeLattice  
     LatticePatch, 86  
 errorKill  
     LatticePatch.cpp, 189  
     LatticePatch.h, 206  
 eval  
     ICSetter, 40  
 exchangeGhostCells  
     LatticePatch, 71  
 f  
     TimeEvolution, 136  
 FLatticeDimensionSet  
     LatticePatch.h, 208  
 FLatticePatchSetUp  
     LatticePatch.h, 208  
 Gauss1D, 11  
     addToSpace, 13  
     Gauss1D, 12  
     kx, 14  
     ky, 14  
     kz, 14  
     phig, 14  
     phix, 14  
     phiy, 15  
     phiz, 15  
     px, 15  
     py, 15  
     pz, 16  
     x0x, 16  
     x0y, 16  
     x0z, 16  
 gauss1Ds  
     ICSetter, 41  
 Gauss2D, 17  
     A1, 20  
     A2, 20  
     addToSpace, 19  
     Amp, 20  
     axis, 20  
     dis, 21  
     Gauss2D, 18  
     lambda, 21  
     Ph0, 21  
     PhA, 21  
     phip, 22  
     w0, 22  
     zr, 22  
 gauss2Ds  
     ICSetter, 41  
 Gauss3D, 23  
     A1, 25  
     A2, 26  
     addToSpace, 25  
     Amp, 26  
     axis, 26  
     dis, 26  
     Gauss3D, 24  
     lambda, 27  
     Ph0, 27  
     PhA, 27  
     phip, 27  
     w0, 28  
     zr, 28  
 gauss3Ds  
     ICSetter, 41  
 gaussian1D, 28  
     k, 29  
     p, 29  
     phi, 29  
     phig, 29  
     x0, 30  
 gaussian2D, 30  
     amp, 31

axis, 31  
ph0, 31  
phA, 31  
phip, 31  
w0, 31  
x0, 32  
zr, 32  
gaussian3D, 32  
amp, 33  
axis, 33  
ph0, 33  
phA, 33  
phip, 33  
w0, 34  
x0, 34  
zr, 34  
gCLData  
    LatticePatch, 86  
gCRData  
    LatticePatch, 86  
generateOutputFolder  
    OutputManager, 97  
generatePatchwork  
    LatticePatch, 82  
    LatticePatch.cpp, 190  
generateTranslocationLookup  
    LatticePatch, 73  
get\_cart\_comm  
    Simulation, 125  
get\_dataPointDimension  
    Lattice, 45  
get\_dx  
    Lattice, 45  
get\_dy  
    Lattice, 46  
get\_dz  
    Lattice, 46  
get\_ghostLayerWidth  
    Lattice, 46  
get\_stencilOrder  
    Lattice, 47  
get\_tot\_lx  
    Lattice, 47  
get\_tot\_ly  
    Lattice, 48  
get\_tot\_lz  
    Lattice, 48  
get\_tot\_noDP  
    Lattice, 49  
get\_tot\_noP  
    Lattice, 49  
get\_tot\_nx  
    Lattice, 49  
get\_tot\_ny  
    Lattice, 50  
get\_tot\_nz  
    Lattice, 50  
getDelta  
    LatticePatch, 75  
getSimCode  
    OutputManager, 98  
ghostCellLeft  
    LatticePatch, 87  
ghostCellLeftToSend  
    LatticePatch, 87  
ghostCellRight  
    LatticePatch, 87  
ghostCellRightToSend  
    LatticePatch, 87  
ghostCells  
    LatticePatch, 88  
ghostCellsToSend  
    LatticePatch, 88  
GhostLayersInitialized  
    LatticePatch.h, 208  
ghostLayerWidth  
    Lattice, 54  
ICSetter, 34  
    add, 35  
    addGauss1D, 36  
    addGauss2D, 37  
    addGauss3D, 37  
    addPlaneWave1D, 38  
    addPlaneWave2D, 39  
    addPlaneWave3D, 39  
    eval, 40  
    gauss1Ds, 41  
    gauss2Ds, 41  
    gauss3Ds, 41  
    planeWaves1D, 42  
    planeWaves2D, 42  
    planeWaves3D, 42  
icsettings  
    Simulation, 133  
ID  
    LatticePatch, 88  
initializeBuffers  
    LatticePatch, 76  
initializeCommunicator  
    Lattice, 50  
initializeCVODEobject  
    Simulation, 125  
initializePatchwork  
    Simulation, 127  
k  
    gaussian1D, 29  
    planewave, 107  
kx  
    Gauss1D, 14  
    PlaneWave, 104  
ky  
    Gauss1D, 14  
    PlaneWave, 104  
kz  
    Gauss1D, 14

PlaneWave, 104  
 lambda  
   Gauss2D, 21  
   Gauss3D, 27  
 Lattice, 43  
   comm, 53  
   dataPointDimension, 53  
   dx, 53  
   dy, 53  
   dz, 53  
   get\_dataPointDimension, 45  
   get\_dx, 45  
   get\_dy, 46  
   get\_dz, 46  
   get\_ghostLayerWidth, 46  
   get\_stencilOrder, 47  
   get\_tot\_lx, 47  
   get\_tot\_ly, 48  
   get\_tot\_lz, 48  
   get\_tot\_noDP, 49  
   get\_tot\_noP, 49  
   get\_tot\_nx, 49  
   get\_tot\_ny, 50  
   get\_tot\_nz, 50  
   ghostLayerWidth, 54  
   initializeCommunicator, 50  
 Lattice, 44  
   my\_prc, 54  
   n\_prc, 54  
   setDiscreteDimensions, 51  
   setPhysicalDimensions, 52  
   statusFlags, 54  
   stencilOrder, 55  
   sunctx, 55  
   tot\_lx, 55  
   tot\_ly, 55  
   tot\_lz, 56  
   tot\_noDP, 56  
   tot\_noP, 56  
   tot\_nx, 56  
   tot\_ny, 57  
   tot\_nz, 57  
 lattice  
   Simulation, 134  
 LatticeDiscreteSetUp  
   SimulationClass.h, 234  
 LatticePatch, 57  
   ~LatticePatch, 61  
   buffData, 84  
   buffX, 84  
   buffY, 84  
   buffZ, 84  
   checkFlag, 61  
   derive, 63  
   derotate, 68  
   discreteSize, 70  
   du, 84  
   duData, 85  
   duLocal, 85  
   dx, 85  
   dy, 85  
   dz, 86  
   envelopeLattice, 86  
   exchangeGhostCells, 71  
   gCLData, 86  
   gCRData, 86  
   generatePatchwork, 82  
   generateTranslocationLookup, 73  
   getDelta, 75  
   ghostCellLeft, 87  
   ghostCellLeftToSend, 87  
   ghostCellRight, 87  
   ghostCellRightToSend, 87  
   ghostCells, 88  
   ghostCellsToSend, 88  
   ID, 88  
   initializeBuffers, 76  
   LatticePatch, 60  
   lgcTox, 88  
   lgcToy, 88  
   lgcToz, 89  
   Llx, 89  
   Lly, 89  
   Llz, 89  
   lx, 90  
   ly, 90  
   lz, 90  
   nx, 90  
   ny, 91  
   nz, 91  
   origin, 76  
   rgcTox, 91  
   rgcToy, 91  
   rgcToz, 92  
   rotateIntoEigen, 77  
   rotateToX, 79  
   rotateToY, 80  
   rotateToZ, 81  
   statusFlags, 92  
   u, 92  
   uAux, 92  
   uAuxData, 93  
   uData, 93  
   uLocal, 93  
   uTox, 93  
   uToy, 94  
   uToz, 94  
   x0, 94  
   xTou, 94  
   y0, 95  
   yTou, 95  
   z0, 95  
   zTou, 95  
 latticePatch  
   Simulation, 134  
 LatticePatch.cpp

check\_error, 188  
check\_retval, 188  
errorKill, 189  
generatePatchwork, 190  
**LatticePatch.h**  
    BuffersInitialized, 207  
    check\_error, 205  
    check\_retval, 205  
    errorKill, 206  
    FLatticeDimensionSet, 208  
    FLatticePatchSetUp, 208  
    GhostLayersInitialized, 208  
    TranslocationLookupSetUp, 208  
**LatticePatchworkSetUp**  
    SimulationClass.h, 234  
**LatticePhysicalSetUp**  
    SimulationClass.h, 234  
**lgcTox**  
    LatticePatch, 88  
**lgcToy**  
    LatticePatch, 88  
**lgcToz**  
    LatticePatch, 89  
**linear1DProp**  
    TimeEvolutionFunctions.cpp, 262  
    TimeEvolutionFunctions.h, 291  
**linear2DProp**  
    TimeEvolutionFunctions.cpp, 264  
    TimeEvolutionFunctions.h, 292  
**linear3DProp**  
    TimeEvolutionFunctions.cpp, 266  
    TimeEvolutionFunctions.h, 294  
**Llx**  
    LatticePatch, 89  
**Lly**  
    LatticePatch, 89  
**Llz**  
    LatticePatch, 89  
**Ix**  
    LatticePatch, 90  
**Iy**  
    LatticePatch, 90  
**Iz**  
    LatticePatch, 90  
**main**  
    main.cpp, 212  
**main.cpp**  
    main, 212  
**my\_prc**  
    Lattice, 54  
**n\_prc**  
    Lattice, 54  
**NLS**  
    Simulation, 134  
**nonlinear1DProp**  
    TimeEvolutionFunctions.cpp, 268  
    TimeEvolutionFunctions.h, 296  
**nonlinear2DProp**  
    TimeEvolutionFunctions.cpp, 273  
    TimeEvolutionFunctions.h, 301  
**nonlinear3DProp**  
    TimeEvolutionFunctions.cpp, 277  
    TimeEvolutionFunctions.h, 305  
**nx**  
    LatticePatch, 90  
**ny**  
    LatticePatch, 91  
**nz**  
    LatticePatch, 91  
**origin**  
    LatticePatch, 76  
**outAllFieldData**  
    Simulation, 128  
**OutputManager**, 96  
    generateOutputFolder, 97  
    getSimCode, 98  
    OutputManager, 97  
    outputStyle, 102  
    outUState, 99  
    Path, 102  
    set\_outputStyle, 101  
    simCode, 102  
    SimCodeGenerator, 101  
**outputManager**  
    Simulation, 134  
**outputStyle**  
    OutputManager, 102  
**outUState**  
    OutputManager, 99  
**p**  
    gaussian1D, 29  
    planewave, 107  
**Path**  
    OutputManager, 102  
**Ph0**  
    Gauss2D, 21  
    Gauss3D, 27  
**ph0**  
    gaussian2D, 31  
    gaussian3D, 33  
**PhA**  
    Gauss2D, 21  
    Gauss3D, 27  
**phA**  
    gaussian2D, 31  
    gaussian3D, 33  
**phi**  
    gaussian1D, 29  
    planewave, 107  
**phig**  
    Gauss1D, 14  
    gaussian1D, 29  
**phip**  
    Gauss2D, 22

Gauss3D, 27  
 gaussian2D, 31  
 gaussian3D, 33  
 phix  
     Gauss1D, 14  
     PlaneWave, 104  
 phiy  
     Gauss1D, 15  
     PlaneWave, 105  
 phiz  
     Gauss1D, 15  
     PlaneWave, 105  
 PlaneWave, 103  
     kx, 104  
     ky, 104  
     kz, 104  
     phix, 104  
     phiy, 105  
     phiz, 105  
     px, 105  
     py, 105  
     pz, 106  
 planewave, 106  
     k, 107  
     p, 107  
     phi, 107  
 PlaneWave1D, 108  
     addToSpace, 109  
     PlaneWave1D, 109  
 PlaneWave2D, 110  
     addToSpace, 112  
     PlaneWave2D, 111  
 PlaneWave3D, 113  
     addToSpace, 115  
     PlaneWave3D, 114  
 planeWaves1D  
     ICSetter, 42  
 planeWaves2D  
     ICSetter, 42  
 planeWaves3D  
     ICSetter, 42  
 px  
     Gauss1D, 15  
     PlaneWave, 105  
 py  
     Gauss1D, 15  
     PlaneWave, 105  
 pz  
     Gauss1D, 16  
     PlaneWave, 106  
 README.md, 139  
 rgcTox  
     LatticePatch, 91  
 rgcToy  
     LatticePatch, 91  
 rgcToz  
     LatticePatch, 92  
 rotateIntoEigen  
     LatticePatch, 77  
 rotateToX  
     LatticePatch, 79  
 rotateToY  
     LatticePatch, 80  
 rotateToZ  
     LatticePatch, 81  
 s10b  
     DerivationStencils.h, 142  
 s10c  
     DerivationStencils.h, 143  
 s10f  
     DerivationStencils.h, 144  
 s11b  
     DerivationStencils.h, 145  
 s11f  
     DerivationStencils.h, 146  
 s12b  
     DerivationStencils.h, 147  
 s12c  
     DerivationStencils.h, 148  
 s12f  
     DerivationStencils.h, 149  
 s13b  
     DerivationStencils.h, 150  
 s13f  
     DerivationStencils.h, 151  
 s1b  
     DerivationStencils.h, 152  
 s1f  
     DerivationStencils.h, 153  
 s2b  
     DerivationStencils.h, 154  
 s2c  
     DerivationStencils.h, 155  
 s2f  
     DerivationStencils.h, 156  
 s3b  
     DerivationStencils.h, 157  
 s3f  
     DerivationStencils.h, 158  
 s4b  
     DerivationStencils.h, 159  
 s4c  
     DerivationStencils.h, 160  
 s4f  
     DerivationStencils.h, 161  
 s5b  
     DerivationStencils.h, 162  
 s5f  
     DerivationStencils.h, 163  
 s6b  
     DerivationStencils.h, 164  
 s6c  
     DerivationStencils.h, 165  
 s6f  
     DerivationStencils.h, 166  
 s7b

DerivationStencils.h, 167  
s7f DerivationStencils.h, 168  
s8b DerivationStencils.h, 169  
s8c DerivationStencils.h, 170  
s8f DerivationStencils.h, 171  
s9b DerivationStencils.h, 172  
s9f DerivationStencils.h, 173  
set\_outputStyle OutputManager, 101  
setDiscreteDimensions Lattice, 51  
setDiscreteDimensionsOfLattice Simulation, 129  
setInitialConditions Simulation, 130  
setPhysicalDimensions Lattice, 52  
setPhysicalDimensionsOfLattice Simulation, 131  
Sim1D SimulationFunctions.cpp, 237  
SimulationFunctions.h, 251  
Sim2D SimulationFunctions.cpp, 240  
SimulationFunctions.h, 254  
Sim3D SimulationFunctions.cpp, 243  
SimulationFunctions.h, 257  
simCode OutputManager, 102  
SimCodeGenerator OutputManager, 101  
Simulation, 115  
~Simulation, 118  
addInitialConditions, 119  
addPeriodicCLayerInX, 120  
addPeriodicCLayerInXY, 120  
advanceToTime, 121  
checkFlag, 122  
checkNoFlag, 124  
cvode\_mem, 133  
get\_cart\_comm, 125  
icsettings, 133  
initializeCVODEobject, 125  
initializePatchwork, 127  
lattice, 134  
latticePatch, 134  
NLS, 134  
outAllFieldData, 128  
outputManager, 134  
setDiscreteDimensionsOfLattice, 129  
setInitialConditions, 130  
setPhysicalDimensionsOfLattice, 131  
Simulation, 117  
start, 132  
statusFlags, 135  
t, 135  
SimulationClass.h  
CvodeObjectSetUp, 233  
LatticeDiscreteSetUp, 234  
LatticePatchworkSetUp, 234  
LatticePhysicalSetUp, 234  
SimulationStarted, 234  
SimulationFunctions.cpp  
Sim1D, 237  
Sim2D, 240  
Sim3D, 243  
timer, 246  
SimulationFunctions.h  
Sim1D, 251  
Sim2D, 254  
Sim3D, 257  
timer, 260  
SimulationStarted  
SimulationClass.h, 234  
src/DerivationStencils.cpp, 139, 140  
src/DerivationStencils.h, 140, 174  
src/ICSetters.cpp, 178  
src/ICSetters.h, 183, 184  
src/LatticePatch.cpp, 187, 192  
src/LatticePatch.h, 203, 209  
src/main.cpp, 212, 220  
src/Outputters.cpp, 224  
src/Outputters.h, 226, 227  
src/SimulationClass.cpp, 228  
src/SimulationClass.h, 232, 235  
src/SimulationFunctions.cpp, 236, 247  
src/SimulationFunctions.h, 250, 261  
src/TimeEvolutionFunctions.cpp, 262, 281  
src/TimeEvolutionFunctions.h, 289, 309  
start  
Simulation, 132  
statusFlags  
Lattice, 54  
LatticePatch, 92  
Simulation, 135  
stencilOrder  
Lattice, 55  
sunctx  
Lattice, 55  
t  
Simulation, 135  
TimeEvolution, 135  
c, 137  
f, 136  
TimeEvolver, 137  
TimeEvolutionFunctions.cpp  
linear1DProp, 262  
linear2DProp, 264  
linear3DProp, 266

nonlinear1DProp, 268  
 nonlinear2DProp, 273  
 nonlinear3DProp, 277  
**TimeEvolutionFunctions.h**  
     linear1DProp, 291  
     linear2DProp, 292  
     linear3DProp, 294  
     nonlinear1DProp, 296  
     nonlinear2DProp, 301  
     nonlinear3DProp, 305  
**TimeEvolver**  
     TimeEvolution, 137  
**timer**  
     SimulationFunctions.cpp, 246  
     SimulationFunctions.h, 260  
**tot\_lx**  
     Lattice, 55  
**tot\_ly**  
     Lattice, 55  
**tot\_lz**  
     Lattice, 56  
**tot\_noDP**  
     Lattice, 56  
**tot\_noP**  
     Lattice, 56  
**tot\_nx**  
     Lattice, 56  
**tot\_ny**  
     Lattice, 57  
**tot\_nz**  
     Lattice, 57  
**TranslocationLookupSetUp**  
     LatticePatch.h, 208

**u**  
     LatticePatch, 92  
**uAux**  
     LatticePatch, 92  
**uAuxData**  
     LatticePatch, 93  
**uData**  
     LatticePatch, 93  
**uLocal**  
     LatticePatch, 93  
**uTox**  
     LatticePatch, 93  
**uToy**  
     LatticePatch, 94  
**uToz**  
     LatticePatch, 94

**w0**  
     Gauss2D, 22  
     Gauss3D, 28  
     gaussian2D, 31  
     gaussian3D, 34

**x0**  
     gaussian1D, 30

    gaussian2D, 32  
     gaussian3D, 34  
     LatticePatch, 94

**x0x**  
     Gauss1D, 16

**x0y**  
     Gauss1D, 16

**x0z**  
     Gauss1D, 16

**xTou**  
     LatticePatch, 94

**y0**  
     LatticePatch, 95

**yTou**  
     LatticePatch, 95

**z0**  
     LatticePatch, 95

**zr**  
     Gauss2D, 22  
     Gauss3D, 28  
     gaussian2D, 32  
     gaussian3D, 34

**zTou**  
     LatticePatch, 95