

VERB_CODE_2.0

Generated by Doxygen 1.5.9

Wed May 26 21:19:29 2010

Contents

1	Main Page	1
2	Todo List	3
3	Namespace Index	7
3.1	Namespace List	7
4	Class Index	9
4.1	Class Hierarchy	9
5	Class Index	11
5.1	Class List	11
6	File Index	13
6.1	File List	13
7	Namespace Documentation	15
7.1	Maths Namespace Reference	15
7.2	Maths::Interpolation Namespace Reference	16
7.3	Output Namespace Reference	17
7.3.1	Function Documentation	17
7.3.1.1	close_log_file	17
7.3.1.2	echo	17
7.3.1.3	open_log_file	18
7.3.1.4	set_output_lvl	18
7.3.2	Variable Documentation	19
7.3.2.1	log_file	19
7.3.2.2	outputLvl	19
7.4	VC Namespace Reference	20
7.4.1	Detailed Description	20

7.5	VF Namespace Reference	21
7.5.1	Detailed Description	22
7.5.2	Function Documentation	22
7.5.2.1	alc	22
7.5.2.2	Alpha2J	23
7.5.2.3	Alpha2Jc	23
7.5.2.4	B	23
7.5.2.5	bounce_time	23
7.5.2.6	density	24
7.5.2.7	Df	24
7.5.2.8	dtostr	24
7.5.2.9	f_interp	25
7.5.2.10	G	25
7.5.2.11	J_L7	25
7.5.2.12	J_L7_corrected	25
7.5.2.13	Jc2Alpha	26
7.5.2.14	Jc_calc	26
7.5.2.15	Kfunc	26
7.5.2.16	max	27
7.5.2.17	mu2pc	27
7.5.2.18	mu_calc	27
7.5.2.19	pc2mu	28
7.5.2.20	pfunc	28
7.5.2.21	Y	28
7.5.2.22	Y_old	29
8	Class Documentation	31
8.1	BoundaryCondition Class Reference	31
8.1.1	Detailed Description	32
8.1.2	Constructor & Destructor Documentation	33
8.1.2.1	BoundaryCondition	33
8.1.2.2	BoundaryCondition	33
8.1.3	Member Function Documentation	33
8.1.3.1	Initialize	33
8.1.3.2	LoadBoundaryCondition	33
8.1.3.3	MakeBoundaryCondition	34
8.1.3.4	operator*	34

8.1.3.5	operator=	35
8.1.3.6	operator=	35
8.1.3.7	Update	35
8.1.4	Member Data Documentation	36
8.1.4.1	BC_parameters	36
8.1.4.2	calculationType	36
8.1.4.3	filename	36
8.1.4.4	initialType	36
8.1.4.5	value	37
8.2	ParamStructure::BoundaryCondition Struct Reference	38
8.2.1	Detailed Description	38
8.2.2	Member Data Documentation	38
8.2.2.1	filename	38
8.2.2.2	type	38
8.2.2.3	value	38
8.3	CalculationMatrix Class Reference	39
8.3.1	Detailed Description	39
8.3.2	Constructor & Destructor Documentation	39
8.3.2.1	CalculationMatrix	39
8.3.2.2	CalculationMatrix	40
8.3.3	Member Function Documentation	40
8.3.3.1	index1d	40
8.3.3.2	Initialize	40
8.3.3.3	writeToFile	41
8.3.4	Member Data Documentation	41
8.3.4.1	change_ind	41
8.3.4.2	initialized	41
8.3.4.3	total_size	41
8.3.4.4	x_size	41
8.3.4.5	y_size	41
8.4	DiffusionCoefficient Class Reference	43
8.4.1	Detailed Description	45
8.4.2	Constructor & Destructor Documentation	45
8.4.2.1	DiffusionCoefficient	45
8.4.2.2	DiffusionCoefficient	46
8.4.2.3	DiffusionCoefficient	46

8.4.3	Member Function Documentation	47
8.4.3.1	AllocateMemory	47
8.4.3.2	Calculate	47
8.4.3.3	Get	48
8.4.3.4	LoadDiffusionCoefficient	49
8.4.3.5	LoadDiffusionCoefficientFromFileWithGrid	50
8.4.3.6	LoadDiffusionCoefficientFromPlaneFile	50
8.4.3.7	MakeDLL	51
8.4.3.8	MakeDLL	51
8.4.3.9	MakeDLL100	51
8.4.3.10	MakeDLL_B	52
8.4.3.11	MakeDLL_BE	52
8.4.3.12	MakeDLL_BE_res	53
8.4.3.13	MakeDLL_FAKE	53
8.4.3.14	operator*	53
8.4.3.15	operator*	53
8.4.3.16	operator+	54
8.4.3.17	operator=	54
8.4.3.18	operator=	54
8.4.3.19	Scale	55
8.4.4	Member Data Documentation	55
8.4.4.1	Dxx_parameters	55
8.4.4.2	is_active	55
8.4.4.3	type	55
8.4.4.4	useScale	55
8.5	DiffusionCoefficientParamStructure Struct Reference	56
8.5.1	Detailed Description	58
8.5.2	Member Function Documentation	58
8.5.2.1	Load_dxx_parameters	58
8.5.3	Member Data Documentation	58
8.5.3.1	Alpha	58
8.5.3.2	Bw	58
8.5.3.3	BwFromLambda	58
8.5.3.4	d_omega	59
8.5.3.5	DxxKp	59
8.5.3.6	DxxName	59

8.5.3.7	DxxType	59
8.5.3.8	EMeV	59
8.5.3.9	eta1	59
8.5.3.10	eta2	59
8.5.3.11	eta3	59
8.5.3.12	f	60
8.5.3.13	filename	60
8.5.3.14	filetype	60
8.5.3.15	L	60
8.5.3.16	lam_max	60
8.5.3.17	lam_min	60
8.5.3.18	loadOrCalculate	60
8.5.3.19	MLT_averaging	61
8.5.3.20	multiplicator	61
8.5.3.21	nint	61
8.5.3.22	nu	61
8.5.3.23	numberDensity	61
8.5.3.24	omega_lc	61
8.5.3.25	Omega_m	61
8.5.3.26	Omega_mType	61
8.5.3.27	omega_uc	62
8.5.3.28	particle	62
8.5.3.29	s	62
8.5.3.30	time_end	62
8.5.3.31	time_start	62
8.5.3.32	useScale	62
8.5.3.33	waveName	62
8.5.3.34	waveType	63
8.6	DiffusionCoefficientsGroup Class Reference	64
8.6.1	Detailed Description	65
8.6.2	Constructor & Destructor Documentation	66
8.6.2.1	DiffusionCoefficientsGroup	66
8.6.2.2	DiffusionCoefficientsGroup	66
8.6.2.3	DiffusionCoefficientsGroup	66
8.6.3	Member Function Documentation	67
8.6.3.1	ActivateAndScale	67

8.6.3.2	Get	67
8.6.3.3	Get	68
8.6.3.4	Get	68
8.6.3.5	Get	68
8.6.3.6	operator=	68
8.6.3.7	operator=	69
8.6.3.8	operator=	69
8.6.4	Member Data Documentation	69
8.6.4.1	DxxList	69
8.7	error_msg Class Reference	70
8.7.1	Detailed Description	70
8.7.2	Constructor & Destructor Documentation	70
8.7.2.1	error_msg	70
8.7.2.2	error_msg	71
8.7.2.3	error_msg	71
8.7.3	Member Function Documentation	71
8.7.3.1	add	71
8.7.3.2	add	71
8.7.3.3	add	71
8.7.3.4	what	72
8.7.4	Member Data Documentation	72
8.7.4.1	errors_stack	72
8.8	ParamStructure::General_Output_parameters Struct Reference	73
8.8.1	Detailed Description	73
8.8.2	Member Data Documentation	73
8.8.2.1	fileName1D	73
8.8.2.2	folderName	73
8.8.2.3	iterStep	74
8.8.2.4	logFileName	74
8.8.2.5	timeStep	74
8.9	Grid Class Reference	75
8.9.1	Detailed Description	76
8.9.2	Constructor & Destructor Documentation	76
8.9.2.1	Grid	76
8.9.2.2	Grid	76
8.9.3	Member Function Documentation	77

8.9.3.1	Initialize	77
8.9.3.2	IsInitialized	78
8.9.3.3	MakeGrid	78
8.9.3.4	Output	79
8.9.4	Member Data Documentation	79
8.9.4.1	alpha	79
8.9.4.2	alphaSize	79
8.9.4.3	epc	80
8.9.4.4	epcSize	80
8.9.4.5	Jacobian	80
8.9.4.6	L	80
8.9.4.7	LSize	80
8.9.4.8	pc	80
8.9.4.9	pcSize	80
8.9.4.10	type	81
8.10	GridElement Class Reference	82
8.10.1	Detailed Description	83
8.10.2	Constructor & Destructor Documentation	83
8.10.2.1	GridElement	83
8.10.2.2	GridElement	83
8.10.2.3	GridElement	83
8.10.3	Member Function Documentation	84
8.10.3.1	AllocateMemory	84
8.10.3.2	Initialize	84
8.10.3.3	Initialize	85
8.10.3.4	Kfunc	85
8.10.3.5	Kfunc	86
8.10.3.6	operator=	86
8.10.3.7	operator=	86
8.10.3.8	SetRegularGridValue	87
8.10.4	Member Data Documentation	87
8.10.4.1	GridElement_parameters	87
8.10.4.2	size	87
8.11	ParamStructure::GridElement Struct Reference	89
8.11.1	Detailed Description	89
8.11.2	Member Data Documentation	89

8.11.2.1	max	89
8.11.2.2	min	89
8.11.2.3	name	90
8.11.2.4	size	90
8.11.2.5	useLogScale	90
8.12	ParamStructure::Interpolation Struct Reference	91
8.12.1	Detailed Description	91
8.12.2	Member Data Documentation	91
8.12.2.1	linearSplineCoef	91
8.12.2.2	maxSecondDerivative	91
8.12.2.3	type	91
8.12.2.4	useLog	92
8.13	ITLIN_DATA Struct Reference	93
8.13.1	Detailed Description	93
8.13.2	Member Enumeration Documentation	93
8.13.2.1	"@0	93
8.13.2.2	"@1	93
8.13.3	Member Data Documentation	94
8.13.3.1	codeid	94
8.13.3.2	mode	94
8.13.3.3	normdx	94
8.13.3.4	res	94
8.13.3.5	residuum	94
8.13.3.6	t	94
8.13.3.7	tau	94
8.14	ITLIN_INFO Struct Reference	95
8.14.1	Detailed Description	95
8.14.2	Member Data Documentation	95
8.14.2.1	iter	95
8.14.2.2	nomatvec	95
8.14.2.3	noprecl	95
8.14.2.4	noprecon	95
8.14.2.5	noprecr	95
8.14.2.6	normdx	96
8.14.2.7	precision	96
8.14.2.8	rcode	96

8.14.2.9	residuum	96
8.14.2.10	subcode	96
8.15	ITLIN_IO Struct Reference	97
8.15.1	Detailed Description	97
8.15.2	Member Data Documentation	97
8.15.2.1	datfile	97
8.15.2.2	datlevel	97
8.15.2.3	errfile	97
8.15.2.4	errlevel	97
8.15.2.5	iterfile	97
8.15.2.6	miscfile	97
8.15.2.7	monfile	98
8.15.2.8	monlevel	98
8.15.2.9	resfile	98
8.16	ITLIN_OPT Struct Reference	99
8.16.1	Detailed Description	99
8.16.2	Member Data Documentation	99
8.16.2.1	convcheck	99
8.16.2.2	datafile	99
8.16.2.3	datalevel	99
8.16.2.4	errorfile	100
8.16.2.5	errorlevel	100
8.16.2.6	i_max	100
8.16.2.7	iterfile	100
8.16.2.8	maxiter	100
8.16.2.9	miscfile	100
8.16.2.10	monitorfile	100
8.16.2.11	monitorlevel	100
8.16.2.12	rescale	100
8.16.2.13	resfile	101
8.16.2.14	rho	101
8.16.2.15	scale	101
8.16.2.16	termcheck	101
8.16.2.17	tol	101
8.17	Maths::Interpolation::Linear Class Reference	102
8.17.1	Detailed Description	102

8.17.2	Constructor & Destructor Documentation	102
8.17.2.1	Linear	102
8.17.2.2	~Linear	102
8.17.3	Member Function Documentation	102
8.17.3.1	getValue	102
8.17.4	Member Data Documentation	103
8.17.4.1	size	103
8.18	Matrix1D< T > Class Template Reference	104
8.18.1	Detailed Description	105
8.18.2	Constructor & Destructor Documentation	106
8.18.2.1	Matrix1D	106
8.18.2.2	Matrix1D	106
8.18.2.3	Matrix1D	106
8.18.2.4	Matrix1D	107
8.18.2.5	~Matrix1D	107
8.18.3	Member Function Documentation	107
8.18.3.1	AllocateMemory	107
8.18.3.2	Interpolate	108
8.18.3.3	max	108
8.18.3.4	maxabs	108
8.18.3.5	operator()	108
8.18.3.6	operator*	108
8.18.3.7	operator*	109
8.18.3.8	operator/	109
8.18.3.9	operator/	109
8.18.3.10	operator=	109
8.18.3.11	operator=	110
8.18.3.12	operator[]	110
8.18.3.13	Polilinear	110
8.18.3.14	Polint	111
8.18.3.15	Ratint	111
8.18.3.16	readFromFile	111
8.18.3.17	readFromFile	112
8.18.3.18	Spline	112
8.18.3.19	writeToFile	113
8.18.3.20	writeToFile	113

8.18.4	Member Data Documentation	113
8.18.4.1	initialized	113
8.18.4.2	name	113
8.18.4.3	size_x	113
8.19	Matrix2D< T > Class Template Reference	115
8.19.1	Detailed Description	116
8.19.2	Constructor & Destructor Documentation	116
8.19.2.1	Matrix2D	116
8.19.2.2	Matrix2D	116
8.19.2.3	Matrix2D	117
8.19.2.4	~Matrix2D	117
8.19.3	Member Function Documentation	117
8.19.3.1	AllocateMemory	117
8.19.3.2	index1d	118
8.19.3.3	Interpolate	118
8.19.3.4	max	118
8.19.3.5	operator*	119
8.19.3.6	operator*	119
8.19.3.7	operator/	119
8.19.3.8	operator/	119
8.19.3.9	operator=	119
8.19.3.10	operator=	120
8.19.3.11	operator[]	120
8.19.3.12	readFromFile	120
8.19.3.13	readFromFile	120
8.19.3.14	writeToFile	121
8.19.3.15	writeToFile	121
8.19.4	Member Data Documentation	121
8.19.4.1	initialized	121
8.19.4.2	name	121
8.19.4.3	size_x	122
8.19.4.4	size_y	122
8.20	Matrix3D< T > Class Template Reference	123
8.20.1	Detailed Description	125
8.20.2	Constructor & Destructor Documentation	125
8.20.2.1	Matrix3D	125

8.20.2.2	Matrix3D	126
8.20.2.3	Matrix3D	126
8.20.2.4	~Matrix3D	126
8.20.3	Member Function Documentation	127
8.20.3.1	abs	127
8.20.3.2	AllocateMemory	127
8.20.3.3	index1d	127
8.20.3.4	max	127
8.20.3.5	maxabs	128
8.20.3.6	operator()	128
8.20.3.7	operator*	128
8.20.3.8	operator*	128
8.20.3.9	operator+	129
8.20.3.10	operator-	129
8.20.3.11	operator/	129
8.20.3.12	operator/	129
8.20.3.13	operator=	129
8.20.3.14	operator=	129
8.20.3.15	operator=	130
8.20.3.16	operator[]	130
8.20.3.17	readFromFile	131
8.20.3.18	readFromFile	131
8.20.3.19	Value	131
8.20.3.20	writeToFile	131
8.20.3.21	writeToFile	131
8.20.3.22	xSlice	132
8.20.3.23	ySlice	132
8.20.3.24	zSlice	132
8.20.4	Member Data Documentation	133
8.20.4.1	change_ind	133
8.20.4.2	initialized	133
8.20.4.3	name	133
8.20.4.4	size_x	133
8.20.4.5	size_y	133
8.20.4.6	size_z	133
8.21	ParamStructure Struct Reference	135

8.21.1 Detailed Description	138
8.21.2 Member Function Documentation	139
8.21.2.1 Load_parameters	139
8.21.3 Member Data Documentation	140
8.21.3.1 alpha_LowerBoundaryCondition	140
8.21.3.2 alpha_UpperBoundaryCondition	140
8.21.3.3 Bf	140
8.21.3.4 constBf	140
8.21.3.5 constKp	140
8.21.3.6 constLpp	140
8.21.3.7 DLLType	140
8.21.3.8 DxxParamStructureList	141
8.21.3.9 fileBf	141
8.21.3.10 fileKp	141
8.21.3.11 fileLpp	141
8.21.3.12 general_Output_parameters	141
8.21.3.13 interpolation	141
8.21.3.14 Kp	141
8.21.3.15 L_LowerBoundaryCondition	142
8.21.3.16 L_UpperBoundaryCondition	142
8.21.3.17 localDiffusionsGrid_alpha	142
8.21.3.18 localDiffusionsGrid_epc	142
8.21.3.19 localDiffusionsGrid_filename	142
8.21.3.20 localDiffusionsGrid_L	142
8.21.3.21 localDiffusionsGrid_pc	142
8.21.3.22 localDiffusionsGrid_type	142
8.21.3.23 Lpp	142
8.21.3.24 nDays	143
8.21.3.25 NoNegative	143
8.21.3.26 outputLvl	143
8.21.3.27 outputModelMatrix	143
8.21.3.28 pc_LowerBoundaryCondition	143
8.21.3.29 pc_UpperBoundaryCondition	143
8.21.3.30 psdLocalDiffusions	143
8.21.3.31 psdRadialDiffusion	143
8.21.3.32 radialDiffusionGrid_alpha	144

8.21.3.33 radialDiffusionGrid_epc	144
8.21.3.34 radialDiffusionGrid_filename	144
8.21.3.35 radialDiffusionGrid_L	144
8.21.3.36 radialDiffusionGrid_pc	144
8.21.3.37 radialDiffusionGrid_type	144
8.21.3.38 sourcesAndLosses	144
8.21.3.39 tau	144
8.21.3.40 tauLpp	144
8.21.3.41 timeStep	145
8.21.3.42 totalIterationsNumber	145
8.21.3.43 useAlphaDifusion	145
8.21.3.44 useBf	145
8.21.3.45 useEnergyAlphaMixedTerms	145
8.21.3.46 useEnergyDifusion	145
8.21.3.47 useKp	145
8.21.3.48 useLpp	146
8.21.3.49 useRadialDifusion	146
8.22 PSD Class Reference	147
8.22.1 Detailed Description	149
8.22.2 Constructor & Destructor Documentation	149
8.22.2.1 PSD	149
8.22.2.2 PSD	150
8.22.2.3 PSD	150
8.22.2.4 PSD	150
8.22.2.5 ~PSD	151
8.22.3 Member Function Documentation	151
8.22.3.1 Diffusion_alpha	151
8.22.3.2 Diffusion_L	152
8.22.3.3 Diffusion_pc	153
8.22.3.4 Diffusion_pc_alpha	154
8.22.3.5 DiffusionMixTermExplicit	156
8.22.3.6 Initialize	157
8.22.3.7 Interpolate	158
8.22.3.8 Load_initial_f	159
8.22.3.9 Load_initial_f	160
8.22.3.10 Load_initial_f_2d	160

8.22.3.11	Load_initial_f_file	161
8.22.3.12	LoadInitialValue	162
8.22.3.13	Output_without_grid	162
8.22.3.14	SourcesAndLosses	163
8.22.4	Member Data Documentation	163
8.22.4.1	output_without_grid_file	163
8.22.4.2	PSD_parameters	163
8.23	ParamStructure::PSD Struct Reference	164
8.23.1	Detailed Description	165
8.23.2	Member Data Documentation	165
8.23.2.1	approximationMethod	165
8.23.2.2	initial_PSD_fileName	165
8.23.2.3	initial_PSD_inner_psd	165
8.23.2.4	initial_PSD_J_L7_function	165
8.23.2.5	initial_PSD_Kp0	165
8.23.2.6	initial_PSD_outer_psd	166
8.23.2.7	initial_PSD_some_constant_value	166
8.23.2.8	initial_PSD_tauSteadyState	166
8.23.2.9	initial_PSD_Type	166
8.23.2.10	output_PSD_fileName4D	166
8.23.2.11	output_PSD_folderName	166
8.23.2.12	output_PSD_timeStep	167
8.23.2.13	SOL_i_max	167
8.23.2.14	SOL_max_iter_err	167
8.23.2.15	SOL_maxiter	167
8.23.2.16	solutionMethod	167
8.24	single_error Class Reference	168
8.24.1	Detailed Description	168
8.24.2	Constructor & Destructor Documentation	168
8.24.2.1	single_error	168
8.24.2.2	single_error	168
8.24.3	Member Data Documentation	168
8.24.3.1	code	168
8.24.3.2	msg	169
8.25	ParamStructure::SL Struct Reference	170
8.25.1	Detailed Description	170

8.25.2	Member Data Documentation	170
8.25.2.1	SL_E_min	170
8.25.2.2	SL_E_min_filename	170
8.25.2.3	SL_L_top	170
8.25.2.4	SL_L_top_filename	170
8.26	SourcesAndLosses Class Reference	171
8.26.1	Detailed Description	171
8.26.2	Constructor & Destructor Documentation	171
8.26.2.1	SourcesAndLosses	171
8.26.2.2	SourcesAndLosses	171
8.26.3	Member Function Documentation	172
8.26.3.1	Initialize	172
8.26.4	Member Data Documentation	172
8.26.4.1	SL_parameters	172
9	File Documentation	175
9.1	bisection.cpp File Reference	175
9.1.1	Detailed Description	175
9.1.2	Function Documentation	176
9.1.2.1	bisection	176
9.2	bisection.d File Reference	177
9.3	bisection.h File Reference	178
9.3.1	Detailed Description	178
9.3.2	Define Documentation	178
9.3.2.1	max_error	178
9.3.3	Function Documentation	178
9.3.3.1	bisection	178
9.4	BoundaryConditions.cpp File Reference	180
9.4.1	Detailed Description	180
9.5	BoundaryConditions.d File Reference	181
9.6	BoundaryConditions.h File Reference	182
9.7	DiffusionCoefficient.cpp File Reference	183
9.7.1	Detailed Description	184
9.7.2	Define Documentation	184
9.7.2.1	double_zero	184
9.7.2.2	min_Dxx	184
9.7.3	Function Documentation	184

9.7.3.1	Alpha_ne	184
9.7.3.2	B	185
9.7.3.3	Daa_root	185
9.7.3.4	Dpa_root	186
9.7.3.5	Dpp_root	186
9.7.3.6	Dxx_ba	187
9.7.3.7	Dxx_local	187
9.7.3.8	f1	188
9.7.3.9	F_cap	188
9.7.3.10	F_cap2	188
9.7.3.11	func_tmp	189
9.7.3.12	int_Daa_loc	189
9.7.3.13	int_Dpa_loc	190
9.7.3.14	int_Dpp_loc	190
9.7.3.15	quad1	191
9.7.3.16	rrouts	191
9.8	DiffusionCoefficient.d File Reference	192
9.9	DiffusionCoefficient.h File Reference	193
9.9.1	Detailed Description	194
9.9.2	Function Documentation	194
9.9.2.1	Alpha_ne	194
9.9.2.2	Daa_root	195
9.9.2.3	Dpa_root	195
9.9.2.4	Dpp_root	196
9.9.2.5	Dxx_ba	196
9.9.2.6	Dxx_local	197
9.9.2.7	f1	197
9.9.2.8	F_cap	197
9.9.2.9	func_tmp	197
9.9.2.10	int_Daa_loc	197
9.9.2.11	int_Dpa_loc	198
9.9.2.12	int_Dpp_loc	198
9.9.2.13	rrouts	199
9.10	erf.cpp File Reference	200
9.10.1	Detailed Description	201
9.10.2	Function Documentation	201

9.10.2.1	<code>bool2str</code>	201
9.10.2.2	<code>erf</code>	201
9.10.2.3	<code>gammln</code>	201
9.10.2.4	<code>gammq</code>	202
9.10.2.5	<code>gammq</code>	202
9.10.2.6	<code>gcf</code>	202
9.10.2.7	<code>gser</code>	203
9.10.2.8	<code>nrerror</code>	203
9.10.2.9	<code>str2bool</code>	204
9.11	<code>erf.d</code> File Reference	205
9.12	<code>erf.h</code> File Reference	206
9.12.1	Detailed Description	207
9.12.2	Define Documentation	207
9.12.2.1	<code>EPS</code>	207
9.12.2.2	<code>FPMIN</code>	207
9.12.2.3	<code>ITMAX</code>	207
9.12.3	Function Documentation	207
9.12.3.1	<code>erf</code>	207
9.12.3.2	<code>gammln</code>	208
9.12.3.3	<code>gammq</code>	208
9.12.3.4	<code>gammq</code>	209
9.12.3.5	<code>gcf</code>	209
9.12.3.6	<code>gser</code>	209
9.13	<code>error.h</code> File Reference	211
9.13.1	Detailed Description	211
9.14	<code>gmres.c</code> File Reference	213
9.14.1	Define Documentation	213
9.14.1.1	<code>MAXITER_DEFAULT</code>	213
9.14.2	Function Documentation	213
9.14.2.1	<code>gmres</code>	213
9.14.2.2	<code>gmres_norm2</code>	214
9.14.2.3	<code>gmres_qrifact</code>	214
9.14.2.4	<code>gmres_qrsolv</code>	215
9.14.3	Variable Documentation	215
9.14.3.1	<code>itlin_ioctl</code>	215
9.15	<code>gmres.d</code> File Reference	216

9.16	Grid.cpp File Reference	217
9.16.1	Detailed Description	217
9.16.2	Define Documentation	218
9.16.2.1	maxERR	218
9.16.3	Function Documentation	218
9.16.3.1	find_alpha	218
9.16.3.2	find_alpha_res	218
9.17	Grid.d File Reference	220
9.18	Grid.h File Reference	221
9.18.1	Detailed Description	222
9.18.2	Function Documentation	222
9.18.2.1	find_alpha	222
9.19	itlin.h File Reference	223
9.19.1	Define Documentation	224
9.19.1.1	DATA	224
9.19.1.2	DATALEVEL	224
9.19.1.3	EPMACH	224
9.19.1.4	ERROR	224
9.19.1.5	ERRORLEVEL	225
9.19.1.6	FITER	225
9.19.1.7	FMISC	225
9.19.1.8	FRES	225
9.19.1.9	MAX	225
9.19.1.10	MIN	225
9.19.1.11	MONITOR	225
9.19.1.12	MONITORLEVEL	225
9.19.1.13	RCODE	225
9.19.1.14	SIGN	226
9.19.1.15	SMALL	226
9.19.2	Typedef Documentation	226
9.19.2.1	MATVEC	226
9.19.2.2	PRECON	226
9.19.3	Enumeration Type Documentation	226
9.19.3.1	CONV_CHECK	226
9.19.3.2	LOGICAL	226
9.19.3.3	PRINT_LEVEL	226

9.19.3.4	TERM_CHECK	227
9.19.4	Function Documentation	227
9.19.4.1	daxpy_	227
9.19.4.2	itlin_dataout	227
9.19.4.3	itlin_noprecon	227
9.19.4.4	itlin_parcheck_and_print	227
9.19.4.5	zibnum_descale	228
9.19.4.6	zibnum_fwalloc	228
9.19.4.7	zibnum_iwalloc	228
9.19.4.8	zibnum_norm2	228
9.19.4.9	zibnum_pfwalloc	228
9.19.4.10	zibnum_scale	228
9.19.4.11	zibnum_scaled_norm2	229
9.19.4.12	zibnum_scaled_sprod	229
9.20	linear.h File Reference	230
9.20.1	Detailed Description	230
9.21	Main.cpp File Reference	231
9.21.1	Detailed Description	232
9.21.2	Define Documentation	232
9.21.2.1	_CRT_SECURE_NO_DEPRECATED	232
9.21.2.2	VERB_VERSION_NUMBER	232
9.21.3	Function Documentation	232
9.21.3.1	check_time	232
9.21.3.2	check_time	232
9.21.3.3	main	233
9.21.4	Variable Documentation	235
9.21.4.1	parameters	235
9.22	Main.d File Reference	236
9.23	Make_boundary.m File Reference	237
9.23.1	Function Documentation	237
9.23.1.1	Bf	237
9.23.1.2	Bf2	237
9.23.1.3	Bf3	237
9.23.1.4	Kp_24_max	237
9.23.2	Variable Documentation	237
9.23.2.1	all	237

9.23.2.2	B_unnorm	237
9.23.2.3	Bf	237
9.23.2.4	Bf1	237
9.23.2.5	Bf3	238
9.23.2.6	dat	238
9.23.2.7	i	238
9.23.2.8	Kp_24_max	238
9.24	Matrix.cpp File Reference	239
9.24.1	Detailed Description	239
9.24.2	Function Documentation	240
9.24.2.1	free_matrix	240
9.24.2.2	free_matrix	240
9.24.2.3	free_matrix	240
9.24.2.4	Linear2D	240
9.24.2.5	matrix	241
9.24.2.6	matrix	241
9.24.2.7	matrix	241
9.25	Matrix.d File Reference	242
9.26	Matrix.h File Reference	243
9.26.1	Detailed Description	244
9.26.2	Typedef Documentation	244
9.26.2.1	DiagMatrix	244
9.27	MatrixSolver.cpp File Reference	245
9.27.1	Detailed Description	247
9.27.2	Define Documentation	247
9.27.2.1	EE	247
9.27.2.2	I	247
9.27.3	Function Documentation	247
9.27.3.1	AddBoundary	247
9.27.3.2	for_norm_A	248
9.27.3.3	gauss_solve	248
9.27.3.4	GetDerivativeVector	248
9.27.3.5	gmres	249
9.27.3.6	gmres_norm2	250
9.27.3.7	gmres_wrapout	250
9.27.3.8	jacobi	251

9.27.3.9	Lapack	251
9.27.3.10	MakeMatrix	252
9.27.3.11	MakeModelMatrix_3D	252
9.27.3.12	matvec	253
9.27.3.13	max2	253
9.27.3.14	mult_vect2	253
9.27.3.15	over_relaxation_diag	254
9.27.3.16	preconr	254
9.27.3.17	SecondDerivativeApproximation_3D	254
9.27.3.18	SolveMatrix	255
9.27.3.19	SOR	255
9.27.3.20	tridag	255
9.28	MatrixSolver.d File Reference	257
9.29	MatrixSolver.h File Reference	258
9.29.1	Detailed Description	259
9.29.2	Function Documentation	260
9.29.2.1	gauss_solve	260
9.29.2.2	gmres_wrapout	260
9.29.2.3	Lapack	261
9.29.2.4	MakeMatrix	262
9.29.2.5	MakeModelMatrix_3D	262
9.29.2.6	over_relaxation_diag	263
9.29.2.7	SecondDerivativeApproximation_3D	264
9.29.2.8	SolveMatrix	264
9.29.2.9	tridag	264
9.30	Output.cpp File Reference	266
9.30.1	Detailed Description	266
9.31	Output.d File Reference	267
9.32	Output.h File Reference	268
9.32.1	Detailed Description	269
9.33	Parameters.cpp File Reference	270
9.33.1	Detailed Description	271
9.33.2	Function Documentation	271
9.33.2.1	bool2str	271
9.33.2.2	load_1d	271
9.33.2.3	ReadFromFile	271

9.33.2.4	str2bool	272
9.33.2.5	StrToVal	272
9.33.2.6	StrToVal	272
9.33.2.7	StrToVal	273
9.33.2.8	StrToVal	273
9.34	Parameters.d File Reference	274
9.35	Parameters.h File Reference	275
9.35.1	Detailed Description	277
9.35.2	Typedef Documentation	277
9.35.2.1	DiffusionCoefficientParamStructureList	277
9.35.3	Function Documentation	277
9.35.3.1	bool2str	277
9.35.3.2	load_1d	277
9.35.3.3	str2bool	278
9.35.3.4	StrToVal	278
9.35.3.5	StrToVal	278
9.35.3.6	StrToVal	278
9.35.3.7	StrToVal	278
9.36	polilinear.cpp File Reference	279
9.36.1	Detailed Description	279
9.36.2	Function Documentation	279
9.36.2.1	polilinear	279
9.37	polilinear.d File Reference	280
9.38	polilinear.h File Reference	281
9.38.1	Detailed Description	281
9.38.2	Function Documentation	281
9.38.2.1	polilinear	281
9.39	polint.cpp File Reference	283
9.39.1	Detailed Description	283
9.39.2	Function Documentation	283
9.39.2.1	polint	283
9.40	polint.d File Reference	285
9.41	polint.h File Reference	286
9.41.1	Detailed Description	286
9.41.2	Function Documentation	286
9.41.2.1	polint	286

9.42	PSD.cpp File Reference	288
9.42.1	Detailed Description	288
9.42.2	Function Documentation	289
9.42.2.1	checkInf	289
9.42.2.2	checkInf	289
9.42.2.3	steady_state	289
9.43	PSD.d File Reference	291
9.44	PSD.h File Reference	292
9.44.1	Detailed Description	293
9.44.2	Function Documentation	293
9.44.2.1	steady_state	293
9.45	ratint.cpp File Reference	294
9.45.1	Detailed Description	294
9.45.2	Define Documentation	294
9.45.2.1	TINY	294
9.45.3	Function Documentation	294
9.45.3.1	ratint	294
9.46	ratint.d File Reference	296
9.47	ratint.h File Reference	297
9.47.1	Detailed Description	297
9.47.2	Function Documentation	297
9.47.2.1	ratint	297
9.48	roots.cpp File Reference	298
9.48.1	Detailed Description	298
9.48.2	Function Documentation	299
9.48.2.1	deflate	299
9.48.2.2	diff_poly	299
9.48.2.3	find_quad	299
9.48.2.4	get_quads	300
9.48.2.5	recurse	300
9.48.2.6	roots	301
9.49	roots.d File Reference	302
9.50	roots.h File Reference	303
9.50.1	Detailed Description	303
9.50.2	Define Documentation	304
9.50.2.1	DBL_EPSILON	304

9.50.2.2	maxiter	304
9.50.3	Function Documentation	304
9.50.3.1	get_quads	304
9.50.3.2	roots	305
9.51	SourcesAndLosses.cpp File Reference	306
9.51.1	Detailed Description	306
9.51.2	Define Documentation	306
9.51.2.1	err	306
9.52	SourcesAndLosses.d File Reference	307
9.53	SourcesAndLosses.h File Reference	308
9.53.1	Detailed Description	308
9.54	spline.cpp File Reference	309
9.54.1	Detailed Description	309
9.54.2	Function Documentation	309
9.54.2.1	spline	309
9.54.2.2	splint	310
9.55	spline.d File Reference	311
9.56	spline.h File Reference	312
9.56.1	Detailed Description	312
9.56.2	Function Documentation	312
9.56.2.1	spline	312
9.56.2.2	splint	313
9.57	utils.c File Reference	315
9.57.1	Define Documentation	315
9.57.1.1	TOLMAX	315
9.57.1.2	TOLMIN	315
9.57.2	Function Documentation	316
9.57.2.1	itlin_dataout	316
9.57.2.2	itlin_noprecon	316
9.57.2.3	itlin_parcheck_and_print	316
9.57.2.4	zibnum_descale	316
9.57.2.5	zibnum_fwalloc	317
9.57.2.6	zibnum_iwalloc	317
9.57.2.7	zibnum_norm2	317
9.57.2.8	zibnum_pfwalloc	317
9.57.2.9	zibnum_scale	317

9.57.2.10	zibnum_scaled_norm2	317
9.57.2.11	zibnum_scaled_sprod	318
9.57.3	Variable Documentation	318
9.57.3.1	itlin_ioctl	318
9.58	utils.d File Reference	319
9.59	variousConstants.h File Reference	320
9.59.1	Detailed Description	320
9.60	variousFunctions.cpp File Reference	321
9.60.1	Detailed Description	322
9.61	variousFunctions.d File Reference	323
9.62	variousFunctions.h File Reference	324
9.62.1	Detailed Description	326

Chapter 1

Main Page

The document [Algorithms.pdf](#) may be useful for understanding the structure of the code. General stuff. We are solving the Fokker-Planck diffusion equation:

$$\frac{\partial f}{\partial t} = \frac{1}{p^2} \frac{\partial}{\partial p} p^2 D_{pp} \frac{\partial f}{\partial p} \Big|_{\text{const } \alpha, L} + \frac{1}{T(\alpha)} \frac{\partial}{\partial \alpha} T(\alpha) D_{\alpha\alpha} \frac{\partial f}{\partial \alpha} \Big|_{\text{const } p, L} + L^2 \frac{\partial}{\partial L} \frac{1}{L^2} D_{LL} \frac{\partial f}{\partial L} \Big|_{\text{const } J_1, J_2}$$

, describes the evolution of phase space density (PSD) in time and phase space. It has three diffusion terms: Radial, Energy, and Pitch-angle. Radial diffusion needs different grid, than two other (local) diffusions. In the code we split the equation into three parts and solve them one after another with interpolation between radial and local grids:

- Radial diffusion
- Interpolation from radial grid to local grid.
- Energy diffusion.
- Pitch-angle diffusion.
- Interpolation from local to radial grid.

We have two grids in the code and phase space density arrays on each of them. PSD values and grids are stored in the class [PSD](#). The structure of the classes is following:

- [PSD](#) - class, holds phase space density values and does diffusions.
- [Grid](#) - grid, in which all that diffusions are.
 - [GridElement](#) L(radial distance), pc (momentum x speed of light in units of MeV) , alpha(pitch angle) , epc(energy in MeV)
- three grid elements (since we have a 3D grid), and forth one, epc, is a function of pc and is just useful enough to keep it.
- [BoundaryCondition](#) upper, lower - upper and lower boundary conditions for diffusion on each grid element

[PSD](#) as general class, contains methods to compute diffusions. It keeps [PSD](#) values in the parent class [Matrix3D](#). [Grid](#) has 3-dimensions, so it has 3 grid elements, and one additional element epc which is just useful for the output. Each grid element is a 3D-Array of values of coordinates. We have to do it this way,

because grids are irregular. [BoundaryCondition](#) class is used for the boundary conditions for diffusion calculation.

[PSD](#) needs diffusion coefficients, grid, and boundary conditions as an input parameter for diffusion functions [PSD::DiffusionL\(\)](#), [PSD::DiffusionPc\(\)](#), [PSD::DiffusionAlpha\(\)](#). Diffusion coefficients for each diffusion are stored inside [DiffusionCoefficientsGroup](#) class. The class structure:

- [DiffusionCoefficientsGroup](#) Daa, Dpcpc etc. - summary of diffusion coefficients for one direction.
 - [DiffusionCoefficient](#) *_chorus, *_hiss, *_EMIC etc - diffusion coefficients by each type of wave separately.

[DiffusionCoefficient](#) s, produced by each wave type, grouped by directions in the [DiffusionCoefficientsGroup](#) class. That class can turn the waves on and off and scale their effect, and store the summation as Daa or Dpp.

Classes constructions. We have empty constructor and functions, like [AllocateMemory\(\)](#) and [Initialize\(\)](#) which define an object. We also have a constructor, which call these functions during execution, just for shortening of the code. And it is like that for almost each class.

Chapter 2

Todo List

Member [DiffusionCoefficient::Scale](#)(double Kp) Upgrade scaling diffusion coefficients procedure with scaling according to external array

Class [DiffusionCoefficientsGroup](#) change DiffusionCoefficientGroup name to the [DiffusionCoefficient](#) name and [DiffusionCoefficient](#) name to something else.

Member [GridElement::Kfunc](#)(GridElement arg) Wired function, should be removed.

Member [Matrix1D::readFromFile](#)(string filename, Matrix1D< T > &grid_x) All matrix readings change to readFromFile() function

Member [Matrix1D::Spline](#)(Matrix1D< T > &old_function, Matrix1D< T > &old_grid, Matrix1D< T > &new_grid, c) Check the index of the last argument of the spline interpolation.

Member [Matrix2D::writeToFile](#)(string filename) Change the name 'writeToFile' in Matrix classes to 'write' or something.

Member [Matrix3D::AllocateMemory](#)(int size_x, int size_y, int size_z) Do not fill initialized matrix with zeros.

Class [SourcesAndLosses](#) Move loss cone losses to the rest of the sources and losses

Member [SourcesAndLosses::Initialize](#)(ParamStructure::SL parameters, [Grid](#) &grid, int numberOfIterations, double t) Add time-array, so loading functions can check time steps and grid.

File [error.h](#) Remove from THE CODE error class, output class, move enclosed classes out, remove variabilities in function collings, change all enum's to strings. In that case, probably, the code can be understood.

Member [main](#) Initialize memory for sources only if we have sources. Actually, I guess we should load sources each time step from a file.

Move writing to the file of 1d parameters somewhere out of main

Member [main](#) Move diffusion coefficients scaling output out of main

Member [free_matrix](#) Move to [Matrix.cpp](#)

Member [free_matrix](#) Move to [Matrix.cpp](#)

Member [free_matrix](#) Move to [Matrix.cpp](#)

Member [matrix](#) Move to [Matrix.cpp](#)

Member [matrix](#) Move to [Matrix.cpp](#)

Member [matrix](#) Move to [Matrix.cpp](#)

File [Output.cpp](#) Rewrite type conversion functions as "any to string" and organize them.
Move all headers to the header files and code to cpp files (if it is possible everywhere).

File [Output.h](#) Rewrite all output with streams insted of old-c functions.

File [PSD.cpp](#) A lot of corrections should be done in [PSD](#) class to make it more logical and less spread.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Maths	15
Maths::Interpolation	16
Output	17
VC (Various constants)	20
VF (Various functions)	21

Chapter 4

Class Index

4.1 Class Hierarchy

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

ParamStructure::BoundaryCondition	38
CalculationMatrix	39
DiffusionCoefficientParamStructure	56
error_msg	70
ParamStructure::General_Output_parameters	73
Grid	75
ParamStructure::GridElement	89
ParamStructure::Interpolation	91
ITLIN_DATA	93
ITLIN_INFO	95
ITLIN_IO	97
ITLIN_OPT	99
Maths::Interpolation::Linear	102
Matrix1D< T >	104
Matrix2D< T >	115
Matrix3D< T >	123
Matrix3D< double >	123
BoundaryCondition	31
DiffusionCoefficient	43
DiffusionCoefficientsGroup	64
GridElement	82
PSD	147
ParamStructure	135
ParamStructure::PSD	164
single_error	168
ParamStructure::SL	170
SourcesAndLosses	171

Chapter 5

Class Index

5.1 Class List

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

BoundaryCondition	31
ParamStructure::BoundaryCondition (Boundary conditions parameters structure)	38
CalculationMatrix (Model matrix (or related matrices) It is based on Diagonal matrix and have methods for conversion from 3D or 2D PSD (and related) arrays into 1d array of unknown elements)	39
DiffusionCoefficient (Class DiffusionCoefficient holds diffusion coefficient matrix and routines to load and calculate it)	43
DiffusionCoefficientParamStructure (Diffusion coefficient parameters)	56
DiffusionCoefficientsGroup (It has DiffusionCoefficient class as a parent class, so it stores there summation of all the coefficients to use in diffusion)	64
error_msg (Error message - stack of single_error s)	70
ParamStructure::General_Output_parameters (General programm output parameters structure)	73
Grid (Computational grid Combined from 3 grid elements: L, pc, Alpha and additional array of epc values for convenience)	75
GridElement (Array of values of coordinate axe)	82
ParamStructure::GridElement (Grid element parameters structure)	89
ParamStructure::Interpolation (Interpolation parameters structure)	91
ITLIN_DATA	93
ITLIN_INFO	95
ITLIN_IO	97
ITLIN_OPT	99
Maths::Interpolation::Linear (Linear interpolation)	102
Matrix1D< T > (Matrix 1D class)	104
Matrix2D< T > (Matrix 2D class)	115
Matrix3D< T > (Matrix 3D class)	123
ParamStructure (Main parameters structure)	135
PSD (Phase Space Density class)	147
ParamStructure::PSD (PSD parameters structure)	164
single_error (Hold some information about error in the code)	168
ParamStructure::SL (Sources and losses)	170
SourcesAndLosses	171

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

bisection.cpp (Bisection method of root finding code)	175
bisection.d	177
bisection.h (Bisection method of root finding header)	178
BoundaryConditions.cpp (Boundary condition class)	180
BoundaryConditions.d	181
BoundaryConditions.h	182
DiffusionCoefficient.cpp (Diffusion coefficients calculation, loading, activating, scaling etc code)	183
DiffusionCoefficient.d	192
DiffusionCoefficient.h (Diffusion coefficients calculation, loading, activating, scaling etc) . . .	193
erf.cpp (Error function)	200
erf.d	205
erf.h (Error function)	206
error.h (Used to work with user's exeptions)	211
gmres.c	213
gmres.d	216
Grid.cpp (Grids, grid elements and boundary conditions source)	217
Grid.d	220
Grid.h (Grids, grid elements and boundary conditions headers)	221
itlin.h	223
linear.h (Linear interpolation)	230
Main.cpp (Main program file)	231
Main.d	236
Make_boundary.m	237
Matrix.cpp	239
Matrix.d	242
Matrix.h (Matrix 1D, 2D and 3D and operations with them)	243
MatrixSolver.cpp (Making model matrixes, solving model matrixes)	245
MatrixSolver.d	257
MatrixSolver.h (Making model matrixes, solving model matrixes)	258
Output.cpp (Logging and screen output)	266
Output.d	267
Output.h (Logging and screen output)	268
Parameters.cpp (Loads parameters)	270

Parameters.d	274
Parameters.h (All parameters, loaded from .ini file are described here in one structure)	275
polilinear.cpp (Polilinear interpolation sources)	279
polilinear.d	280
polilinear.h (Polilinear interpolation headers)	281
polint.cpp (Some other interpolation from numerical recepies)	283
polint.d	285
polint.h (Some other interpolation from numerical recepies)	286
PSD.cpp (Makes operations with PSD (like, diffusion))	288
PSD.d	291
PSD.h (Phase Space Density (PSD))	292
ratint.cpp (Some other interpolation)	294
ratint.d	296
ratint.h (Some other interpolation)	297
rroots.cpp (Finds all roots of polynomial by first finding quadratic factors using Bairstow's method, then extracting roots from quadratics)	298
rroots.d	302
rroots.h (Finds all roots of polynomial by first finding quadratic factors using Bairstow's method, then extracting roots from quadratics)	303
SourcesAndLosses.cpp (Sources and losses calculation)	306
SourcesAndLosses.d	307
SourcesAndLosses.h (Sources and losses calculation)	308
spline.cpp (Spline interpolation)	309
spline.d	311
spline.h (Spline interpolation)	312
utils.c	315
utils.d	319
variousConstants.h (Variose usefull constants)	320
variousFunctions.cpp (Variose functions)	321
variousFunctions.d	323
variousFunctions.h (Variose functions)	324

Chapter 7

Namespace Documentation

7.1 Maths Namespace Reference

Namespaces

- namespace [Interpolation](#)

7.2 Maths::Interpolation Namespace Reference

Classes

- class [Linear](#)
Linear interpolation.

7.3 Output Namespace Reference

Functions

- void [open_log_file](#) (string filename)
- void [close_log_file](#) ()
- void [set_output_lvl](#) (int new_outputLvl)
- void [echo](#) (int msglvl, const char *format,...)

Variables

- ofstream * [log_file](#)
- int [outputLvl](#) = 0

7.3.1 Function Documentation

7.3.1.1 void Output::close_log_file ()

Definition at line 29 of file Output.cpp.

Referenced by main().

Here is the caller graph for this function:



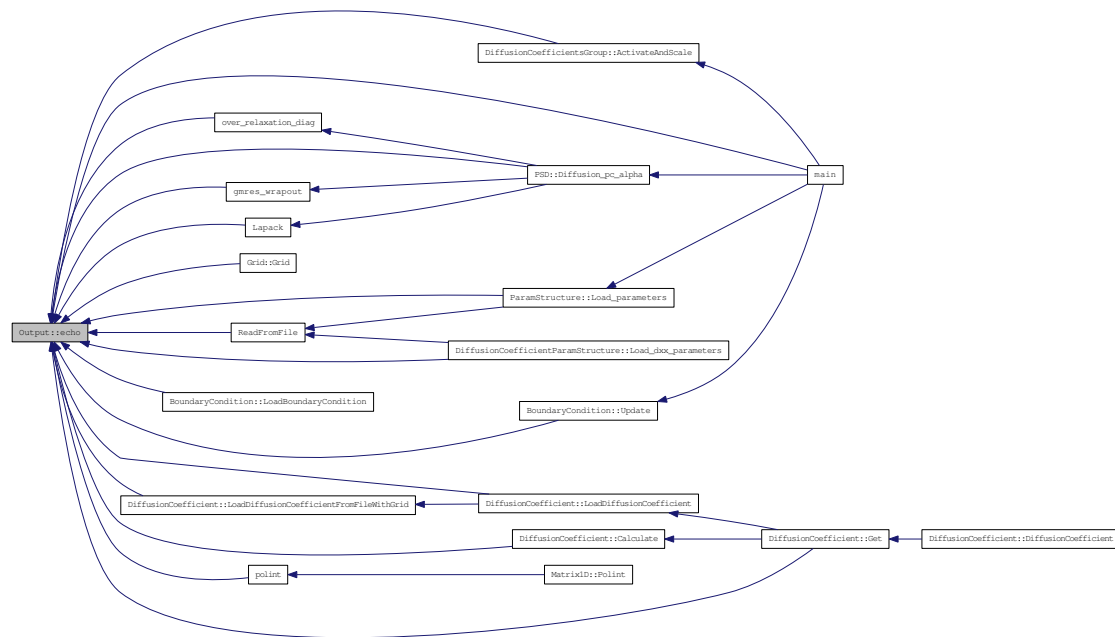
7.3.1.2 void Output::echo (int msglvl, const char *format, ...)

Definition at line 37 of file Output.cpp.

References outputLvl.

Referenced by DiffusionCoefficientsGroup::ActivateAndScale(), DiffusionCoefficient::Calculate(), PSD::Diffusion_pc_alpha(), DiffusionCoefficient::Get(), gmres_wrapout(), Grid::Grid(), Lapack(), DiffusionCoefficientParamStructure::Load_dxx_parameters(), ParamStructure::Load_parameters(), BoundaryCondition::LoadBoundaryCondition(), DiffusionCoefficient::LoadDiffusionCoefficient(), DiffusionCoefficient::LoadDiffusionCoefficientFromFileWithGrid(), main(), over_relaxation_diag(), polint(), ReadFromFile(), and BoundaryCondition::Update().

Here is the caller graph for this function:



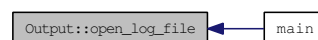
7.3.1.3 void Output::open_log_file (string filename)

Definition at line 20 of file Output.cpp.

References log_file.

Referenced by main().

Here is the caller graph for this function:



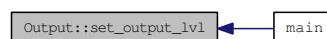
7.3.1.4 void Output::set_output_lvl (int new_outputLvl)

Definition at line 33 of file Output.cpp.

References outputLvl.

Referenced by main().

Here is the caller graph for this function:



7.3.2 Variable Documentation

7.3.2.1 `ofstream* Output::log_file`

Definition at line 16 of file Output.cpp.

Referenced by `open_log_file()`.

7.3.2.2 `int Output::outputLvl = 0`

Definition at line 17 of file Output.cpp.

Referenced by `echo()`, and `set_output_lvl()`.

7.4 VC Namespace Reference

Various constants.

7.4.1 Detailed Description

Various constants.

7.5 VF Namespace Reference

Various functions.

Functions

- double [G](#) (double x)
G-function ;-).
- double [B](#) (double Lparam)
Computation of dipole magnetic field.
- double [Df](#) (double L, double Kp)
Radial Diffusion coeficeint computed following [Brautigam and Albet , 2000].
- double [pfunc](#) (double K)
Computation of moumentum from Kinetic energy.
- double [Kfunc](#) (double pc)
Computation of Kinetic energy from given momentum.
- double [bounce_time](#) (double E, double L)
Bounce time.
- double [J_L7](#) (double K, double x1, double y1, double x2, double y2)
Specifying outer boundary.
- double [J_L7_corrected](#) (double K)
Specifying outer boundary, more accurate function.
- double [mu2pc](#) (double L, double mu, double alpha)
Calculating pc from L, mu, alpha.
- double [pc2mu](#) (double L, double pc, double alpha)
Calculating mu from L, pc, alpha.
- double [Alpha2J](#) (double L, double pc, double Alpha)
Calculating J from L, pc, Alpha.
- double [alc](#) (double L)
Lost cone in rad.
- double [f_interp](#) (double E, double f1, double E1, double f2, double E2)
Getting f on specific energy.
- double [mu_calc](#) (double L, double pc, double Alpha)
Calutulating mu from L, pc, Alpha.
- double [Jc_calc](#) (double L, double pc, double Alpha)

Calculating J from L, pc, Alpha.

- double [Y_old](#) (double alpha)
Y(alpha) function.
- double [Y](#) (double alpha)
Y(alpha) function.
- double [density](#) (double L)
Chorus density model?
- double [max](#) (double v1, double v2)
Chorus density model?
- string [dtostr](#) (double n)
Double to string.
- double [Jc2Alpha](#) (double L, double pc, double Jc)
- double [Alpha2Jc](#) (double L, double pc, double Alpha)

7.5.1 Detailed Description

Various functions.

7.5.2 Function Documentation

7.5.2.1 double VF::alc (double L)

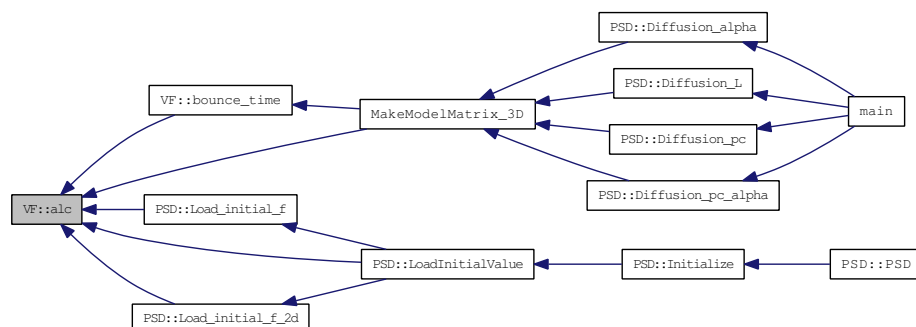
Lost cone in rad.

$$\sin^2(a) = L^2(-3)/(4 - 3/L)^{(1/2)}$$

Definition at line 123 of file variousFunctions.cpp.

Referenced by bounce_time(), PSD::Load_initial_f(), PSD::Load_initial_f_2d(), PSD::LoadInitialValue(), and MakeModelMatrix_3D().

Here is the caller graph for this function:



7.5.2.2 double VF::Alpha2J (double *L*, double *pc*, double *Alpha*)

Calculating J from L, pc, Alpha.

Definition at line 114 of file variousFunctions.cpp.

7.5.2.3 double VF::Alpha2Jc (double *L*, double *pc*, double *Alpha*)

7.5.2.4 double VF::B (double *Lparam*)

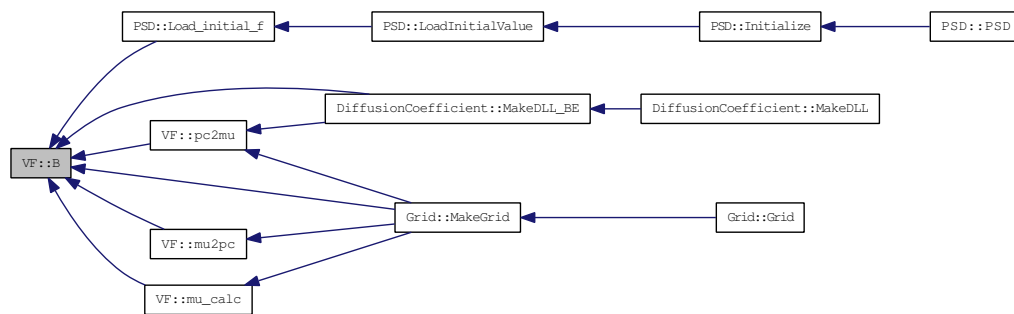
Computation of dipole magnetic field.

in Gauss

Definition at line 27 of file variousFunctions.cpp.

Referenced by PSD::Load_initial_f(), DiffusionCoefficient::MakeDLL_BE(), Grid::MakeGrid(), mu2pc(), mu_calc(), and pc2mu().

Here is the caller graph for this function:



7.5.2.5 double VF::bounce_time (double *E*, double *L*)

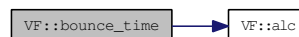
Bounce time.

Definition at line 51 of file variousFunctions.cpp.

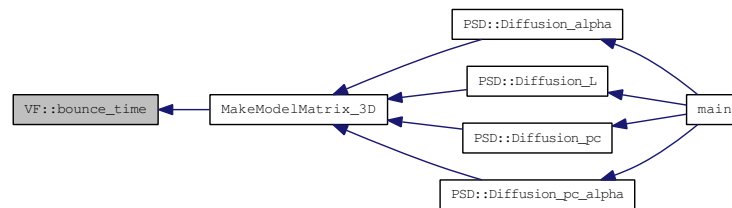
References alc().

Referenced by MakeModelMatrix_3D().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.2.6 double VF::density (double L)

Chorus density model?

Definition at line 177 of file variousFunctions.cpp.

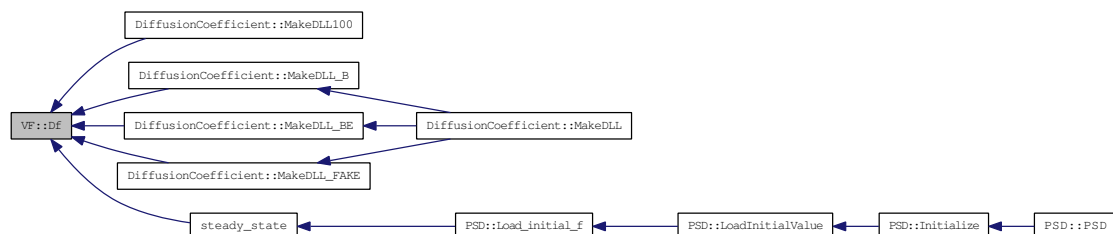
7.5.2.7 double VF::Df (double L , double Kp)

Radial Diffusion coeficeint computed following [Brautigam and Albet , 2000].

Definition at line 32 of file variousFunctions.cpp.

Referenced by DiffusionCoefficient::MakeDLL100(), DiffusionCoefficient::MakeDLL_B(), DiffusionCoefficient::MakeDLL_BE(), DiffusionCoefficient::MakeDLL_FAKE(), and steady_state().

Here is the caller graph for this function:



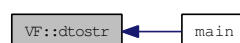
7.5.2.8 string VF::dtostr (double n)

Double to string.

Definition at line 200 of file variousFunctions.cpp.

Referenced by main().

Here is the caller graph for this function:



7.5.2.9 double VF::f_interp (double *E*, double *f1*, double *E1*, double *f2*, double *E2*)

Getting *f* on specific energy.

Linear interpolation of logarithm.

Definition at line 130 of file variousFunctions.cpp.

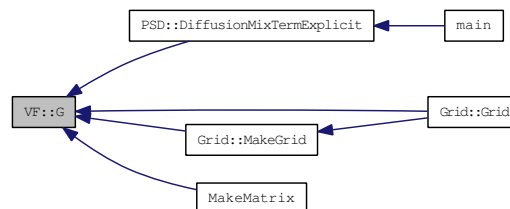
7.5.2.10 double VF::G (double *x*)

G-function ;-).

Definition at line 19 of file variousFunctions.cpp.

Referenced by PSD::DiffusionMixTermExplicit(), Grid::Grid(), Grid::MakeGrid(), and MakeMatrix().

Here is the caller graph for this function:

**7.5.2.11 double VF::J_L7 (double *K*, double *x1*, double *y1*, double *x2*, double *y2*)**

Specifying outer boundary.

Definition at line 63 of file variousFunctions.cpp.

Referenced by `J_L7_corrected()`, and `PSD::Load_initial_f()`.

Here is the caller graph for this function:

**7.5.2.12 double VF::J_L7_corrected (double *K*)**

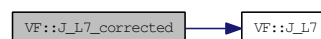
Specifying outer boundary, more accurate function.

Definition at line 80 of file variousFunctions.cpp.

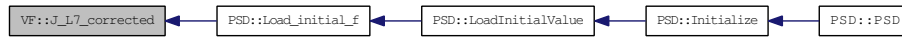
References `i`, and `J_L7()`.

Referenced by `PSD::Load_initial_f()`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.2.13 double VF::Jc2Alpha (double L , double pc , double Jc)

7.5.2.14 double VF::Jc_calc (double L , double pc , double $Alpha$)

Calculating J from L, pc, Alpha.

Definition at line 152 of file variousFunctions.cpp.

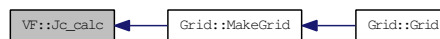
References Y().

Referenced by Grid::MakeGrid().

Here is the call graph for this function:



Here is the caller graph for this function:



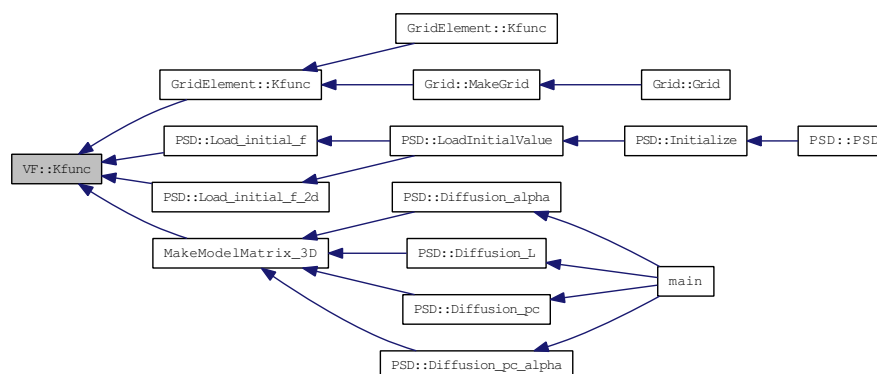
7.5.2.15 double VF::Kfunc (double pc)

Computation of Kinetic energy from given momentum.

Definition at line 46 of file variousFunctions.cpp.

Referenced by GridElement::Kfunc(), PSD::Load_initial_f(), PSD::Load_initial_f_2d(), and MakeModelMatrix_3D().

Here is the caller graph for this function:



7.5.2.16 double VF::max (double *v1*, double *v2*)

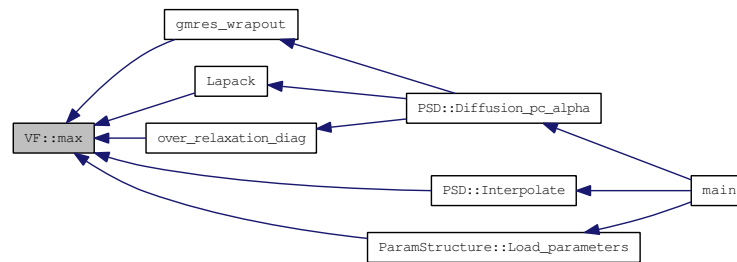
Chorus density model?

Return maximum.

Definition at line 195 of file variousFunctions.cpp.

Referenced by gmres_wrapout(), PSD::Interpolate(), Lapack(), ParamStructure::Load_parameters(), and over_relaxation_diag().

Here is the caller graph for this function:



7.5.2.17 double VF::mu2pc (double *L*, double *mu*, double *alpha*)

Calculating pc from L, mu, alpha.

Definition at line 94 of file variousFunctions.cpp.

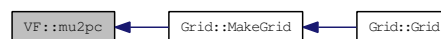
References B().

Referenced by Grid::MakeGrid().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.2.18 double VF::mu_calc (double *L*, double *pc*, double *Alpha*)

Calculating mu from L, pc, Alpha.

Definition at line 145 of file variousFunctions.cpp.

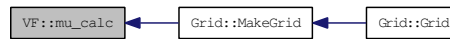
References B().

Referenced by Grid::MakeGrid().

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.2.19 double VF::pc2mu (double *L*, double *pc*, double *alpha*)

Calculating mu from L, pc, alpha.

Definition at line 102 of file variousFunctions.cpp.

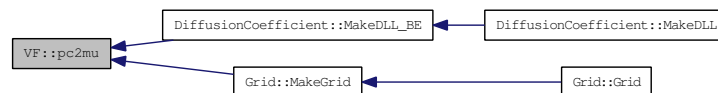
References B().

Referenced by DiffusionCoefficient::MakeDLL_BE(), and Grid::MakeGrid().

Here is the call graph for this function:



Here is the caller graph for this function:



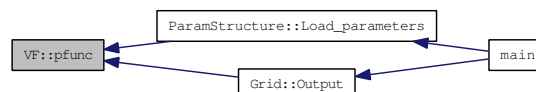
7.5.2.20 double VF::pfunc (double *K*)

Computation of moumentum from Kinetic energy.

Definition at line 39 of file variousFunctions.cpp.

Referenced by ParamStructure::Load_parameters(), and Grid::Output().

Here is the caller graph for this function:



7.5.2.21 double VF::Y (double *alpha*)

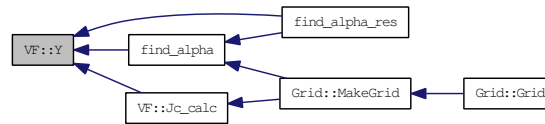
Y(alpha) function.

More accurate approximation.

Definition at line 167 of file variousFunctions.cpp.

Referenced by find_alpha(), find_alpha_res(), and Jc_calc().

Here is the caller graph for this function:



7.5.2.22 double VF::Y_old (double *alpha*)

Y(alpha) function.

Approximation.

Definition at line 159 of file variousFunctions.cpp.

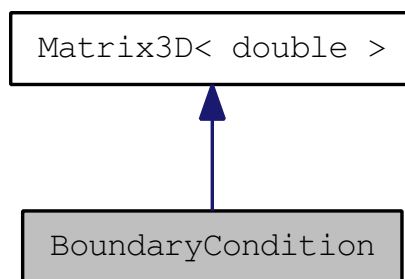
Chapter 8

Class Documentation

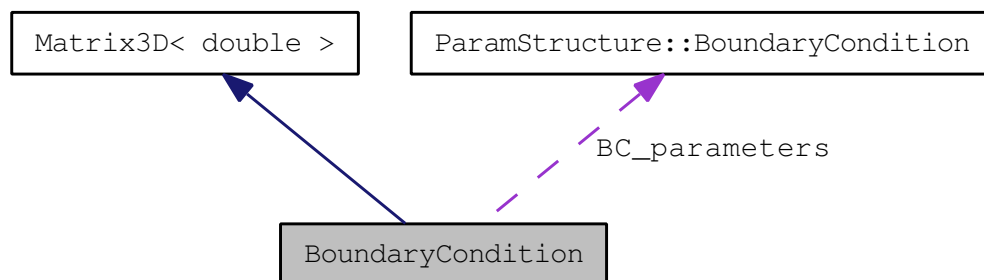
8.1 BoundaryCondition Class Reference

```
#include <BoundaryConditions.h>
```

Inheritance diagram for BoundaryCondition:



Collaboration diagram for BoundaryCondition:



Public Member Functions

- [BoundaryCondition\(\)](#)

Default constructor - do nothing.

- [BoundaryCondition](#) (int time_size, int size_two, int size_three, [ParamStructure::BoundaryCondition](#) parameters)
Constructor.
- void [Initialize](#) ([ParamStructure::BoundaryCondition](#) parameters)
Boundary condition initialization.
- [BoundaryCondition](#) & [operator=](#) (double val)
Makes boundary value equal to val.
- [BoundaryCondition](#) & [operator=](#) ([Matrix3D](#)< double > M)
Make boundary value equal to Matrix M.
- [BoundaryCondition](#) [operator*](#) (double val)
Multiply boundary to a value val;.
- void [MakeBoundaryCondition](#) ([Matrix2D](#)< double > psd2DSlice, [Matrix2D](#)< double > gridElement1, [Matrix2D](#)< double > gridElement2)
Making boundary conditions.
- void [LoadBoundaryCondition](#) ([Matrix2D](#)< double > gridElement1, [Matrix2D](#)< double > gridElement2)
Load boundary conditions from file.
- void [Update](#) (int it, [Matrix2D](#)< double > PSD_2D_Slice)
Update boundary condition for current time step.

Public Attributes

- [ParamStructure::BoundaryCondition](#) BC_parameters
- string [calculationType](#)
Type: constant function/constant derivative.
- string [initialType](#)
Initial type: how to set up boundary conditions. (Like read from a file, etc).
- double [value](#)
Base value at the boundary.
- string [filename](#)
File name for loading of additional sources/losses at the boundary.

8.1.1 Detailed Description

Definition at line 21 of file BoundaryConditions.h.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 BoundaryCondition::BoundaryCondition () [inline]

Default constructor - do nothing.

Definition at line 39 of file BoundaryConditions.h.

8.1.2.2 BoundaryCondition::BoundaryCondition (int *time_size*, int *size_two*, int *size_three*, ParamStructure::BoundaryCondition *parameters*)

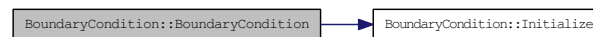
Constructor.

Run constructor of the parent matrix class.

Definition at line 11 of file BoundaryConditions.cpp.

References Initialize().

Here is the call graph for this function:



8.1.3 Member Function Documentation

8.1.3.1 void BoundaryCondition::Initialize (ParamStructure::BoundaryCondition *parameters*)

Boundary condition initialization.

Saving type of BC (on value/derivative etc), readed from parameters file, to the class variable "calculation-Type". Saving value on the BC, readed from file, to the class variable "value".

Parameters:

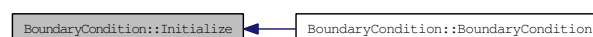
ParamStructure::BoundaryCondition parameters - boundary condition parameters structure

Definition at line 24 of file BoundaryConditions.cpp.

References BC_parameters, calculationType, ParamStructure::BoundaryCondition::filename, filename, initialType, ParamStructure::BoundaryCondition::type, ParamStructure::BoundaryCondition::value, and value.

Referenced by BoundaryCondition().

Here is the caller graph for this function:



8.1.3.2 void BoundaryCondition::LoadBoundaryCondition (Matrix2D< double > *gridElement1*, Matrix2D< double > *gridElement2*)

Load boundary conditions from file.

Here we actually fill-out 2D array of boundary conditions. (in general, 3D grid, case)

Parameters:

Matrix2D<double> psd2DSlice - 2D array, slice of [PSD](#) on a boundary

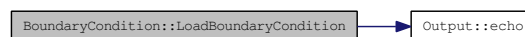
gridElement1 - one side of the grid

gridElement2 - second side of the grid

Definition at line 119 of file BoundaryConditions.cpp.

References `Output::echo()`, `err`, `filename`, `initialType`, `Matrix3D< double >::size_x`, `Matrix3D< double >::size_y`, and `Matrix3D< double >::size_z`.

Here is the call graph for this function:



8.1.3.3 void BoundaryCondition::MakeBoundaryCondition (Matrix2D< double > psd2DSlice, Matrix2D< double > gridElement1, Matrix2D< double > gridElement2)

Making boundary conditions.

Here we actually fill-out 2D array of boundary conditions. (in general, 3D grid case)

Parameters:

Matrix2D<double> psd2DSlice - 2D array, slice of [PSD](#) on a boundary

gridElement1 - one side of the grid

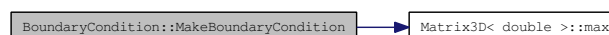
gridElement2 - second side of the grid

Definition at line 78 of file BoundaryConditions.cpp.

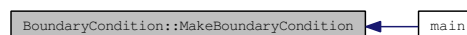
References `calculationType`, `initialType`, `Matrix3D< double >::max()`, `Matrix3D< double >::size_x`, `Matrix3D< double >::size_y`, `Matrix3D< double >::size_z`, and `value`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.3.4 BoundaryCondition BoundaryCondition::operator* (double val)

Multiply boundary to a value val;

Parameters:

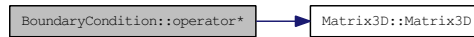
double val - value val.

Reimplemented from [Matrix3D< double >](#).

Definition at line 215 of file BoundaryConditions.cpp.

References [Matrix3D< T >::Matrix3D\(\)](#).

Here is the call graph for this function:



8.1.3.5 BoundaryCondition & BoundaryCondition::operator= (Matrix3D< double > M)

Make boundary value equal to Matrix M.

Parameters:

Matrix2D<double> M - matrix M

Definition at line 205 of file BoundaryConditions.cpp.

References `operator=()`.

Here is the call graph for this function:



8.1.3.6 BoundaryCondition & BoundaryCondition::operator= (double val)

Makes boundary value equal to val.

Parameters:

double val - value val;

Reimplemented from [Matrix3D< double >](#).

Definition at line 194 of file BoundaryConditions.cpp.

Referenced by `operator=()`.

Here is the caller graph for this function:



8.1.3.7 void BoundaryCondition::Update (int iteration, Matrix2D< double > PSD_2D_Slice)

Update boundary condition for current time step.

Parameters:

iteration - iteration number

PSD_2D_Slice - 2d slice of [PSD](#) for that BC

Definition at line 172 of file BoundaryConditions.cpp.

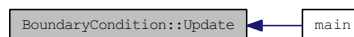
References [Output::echo\(\)](#), [initialType](#), [Matrix2D< T >::name](#), [Matrix3D< double >::size_y](#), and [Matrix3D< double >::size_z](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.1.4 Member Data Documentation

8.1.4.1 ParamStructure::BoundaryCondition BoundaryCondition::BC_parameters

Definition at line 26 of file BoundaryConditions.h.

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

8.1.4.2 string BoundaryCondition::calculationType

Type: constant function/constant derivative.

Definition at line 29 of file BoundaryConditions.h.

Referenced by [Initialize\(\)](#), [main\(\)](#), and [MakeBoundaryCondition\(\)](#).

8.1.4.3 string BoundaryCondition::filename

File name for loading of additional sources/losses at the boundary.

Definition at line 35 of file BoundaryConditions.h.

Referenced by [Initialize\(\)](#), and [LoadBoundaryCondition\(\)](#).

8.1.4.4 string BoundaryCondition::initialType

Initial type: how to set up boundary conditions. (Like read from a file, etc).

Definition at line 31 of file BoundaryConditions.h.

Referenced by [Initialize\(\)](#), [LoadBoundaryCondition\(\)](#), [MakeBoundaryCondition\(\)](#), [PSD::PSD\(\)](#), and [Update\(\)](#).

8.1.4.5 double BoundaryCondition::value

Base value at the boundary.

Definition at line 33 of file BoundaryConditions.h.

Referenced by Initialize(), and MakeBoundaryCondition().

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

- [BoundaryConditions.h](#)
- [BoundaryConditions.cpp](#)

8.2 ParamStructure::BoundaryCondition Struct Reference

Boundary conditions parameters structure.

```
#include <Parameters.h>
```

Public Attributes

- string [type](#)
Type. Check StrToVal(string input, BoundaryConditionTypes &place) for known values.
- double [value](#)
Value (if it is constant value on the boundary).
- string [filename](#)
File name to load boundary conditions from.

8.2.1 Detailed Description

Boundary conditions parameters structure.

Definition at line 150 of file Parameters.h.

8.2.2 Member Data Documentation

8.2.2.1 string ParamStructure::BoundaryCondition::filename

File name to load boundary conditions from.

Definition at line 156 of file Parameters.h.

Referenced by BoundaryCondition::Initialize(), and ParamStructure::Load_parameters().

8.2.2.2 string ParamStructure::BoundaryCondition::type

Type. Check StrToVal(string input, BoundaryConditionTypes &place) for known values.

Definition at line 152 of file Parameters.h.

Referenced by BoundaryCondition::Initialize(), and ParamStructure::Load_parameters().

8.2.2.3 double ParamStructure::BoundaryCondition::value

Value (if it is constant value on the boundary).

Definition at line 154 of file Parameters.h.

Referenced by BoundaryCondition::Initialize(), and ParamStructure::Load_parameters().

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

- [Parameters.h](#)

8.3 CalculationMatrix Class Reference

Model matrix (or related matrices) It is based on Diagonal matrix and have methods for conversion from 3D or 2D **PSD** (and related) arrays into 1d array of unknown elements.

```
#include <Matrix.h>
```

Public Member Functions

- [CalculationMatrix](#) ()
- [CalculationMatrix](#) (int [x_size](#), int [y_size](#)=1, int [z_size](#)=1, int [n_of_diags](#)=1)
Constructor for [CalculationMatrix](#) class.
- void [Initialize](#) (int [x_size](#), int [y_size](#)=1, int [z_size](#)=1, int [n_of_diags](#)=1)
Allocating memory for [CalculationMatrix](#).
- int [index1d](#) (int x, int y=0, int z=0)
Function returns 1d index for 2D-3D arrays.
- void [writeToFile](#) (string filename)
Save matrix to file.

Public Attributes

- bool [initialized](#)
- int [x_size](#)
- int [y_size](#)
- int [total_size](#)
- string [change_ind](#)
Variables useful for changes tracking (store here time when changed).

8.3.1 Detailed Description

Model matrix (or related matrices) It is based on Diagonal matrix and have methods for conversion from 3D or 2D **PSD** (and related) arrays into 1d array of unknown elements.

Definition at line 218 of file Matrix.h.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 CalculationMatrix::CalculationMatrix () [inline]

Definition at line 227 of file Matrix.h.

References initialized.

8.3.2.2 CalculationMatrix::CalculationMatrix (int *x_size*, int *y_size* = 1, int *z_size* = 1, int *n_of_diags* = 1)

Constructor for [CalculationMatrix](#) class.

Parameters:

x_size - x size

y_size - y size

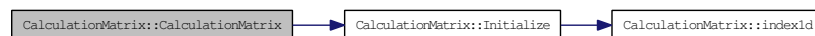
z_size - z size

n_of_diags - NUMBER OF DIAGONALS ABOVE THE MAIN DIAGONAL (main diagonal is not counted)

Definition at line 1579 of file Matrix.cpp.

References Initialize(), and initialized.

Here is the call graph for this function:



8.3.3 Member Function Documentation

8.3.3.1 int CalculationMatrix::index1d (int *x*, int *y* = 0, int *z* = 0)

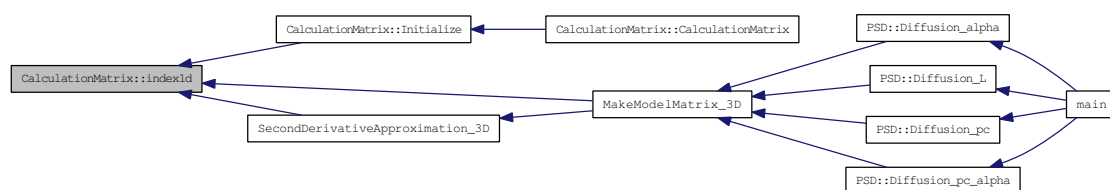
Function returns 1d index for 2D-3D arrays.

Definition at line 1644 of file Matrix.cpp.

References *x_size*, and *y_size*.

Referenced by Initialize(), MakeModelMatrix_3D(), and SecondDerivativeApproximation_3D().

Here is the caller graph for this function:



8.3.3.2 void CalculationMatrix::Initialize (int *x_size*, int *y_size* = 1, int *z_size* = 1, int *n_of_diags* = 1)

Allocating memory for [CalculationMatrix](#).

Definition at line 1587 of file Matrix.cpp.

References index1d(), initialized, and total_size.

Referenced by CalculationMatrix().

Here is the call graph for this function:



Here is the caller graph for this function:



8.3.3.3 void CalculationMatrix::writeToFile (string *filename*)

Save matrix to file.

Definition at line 1651 of file Matrix.cpp.

References `total_size`.

8.3.4 Member Data Documentation

8.3.4.1 string CalculationMatrix::change_ind

Variables useful for changes tracking (store here time when changed).

Definition at line 224 of file Matrix.h.

Referenced by `MakeModelMatrix_3D()`.

8.3.4.2 bool CalculationMatrix::initialized

Definition at line 221 of file Matrix.h.

Referenced by `CalculationMatrix()`, and `Initialize()`.

8.3.4.3 int CalculationMatrix::total_size

Definition at line 222 of file Matrix.h.

Referenced by `Initialize()`, and `writeToFile()`.

8.3.4.4 int CalculationMatrix::x_size

Definition at line 222 of file Matrix.h.

Referenced by `indexId()`.

8.3.4.5 int CalculationMatrix::y_size

Definition at line 222 of file Matrix.h.

Referenced by `indexId()`.

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

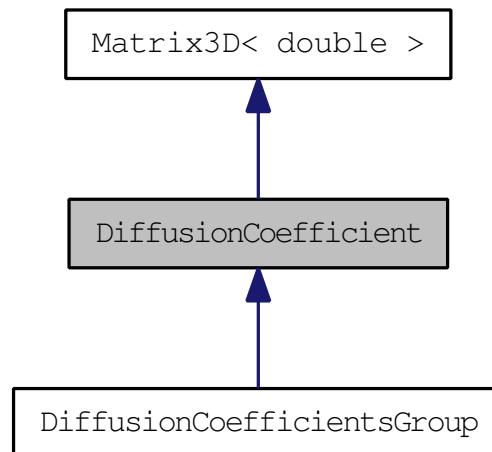
- [Matrix.h](#)
- [Matrix.cpp](#)

8.4 DiffusionCoefficient Class Reference

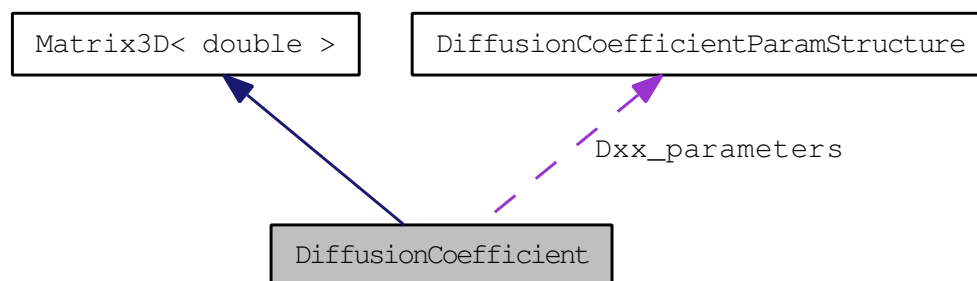
Class [DiffusionCoefficient](#) holds diffusion coefficient matrix and routines to load and calculate it.

```
#include <DiffusionCoefficient.h>
```

Inheritance diagram for DiffusionCoefficient:



Collaboration diagram for DiffusionCoefficient:



Public Member Functions

- `double Scale (double Kp)`
Function, that scale diffusion coefficients.
- `DiffusionCoefficient (int L_size, int pc_size, int alpha_size)`
Constructor. Also runs parent class constructur `Matrix3D<double>(L_size, pc_size, alpha_size)` and it's actually all it does.
- `DiffusionCoefficient (Grid &grid)`
Constructor.
- `DiffusionCoefficient (Grid &grid, DiffusionCoefficientParamStructure DxxParamStructure)`
Constructor.

- void [AllocateMemory](#) (int L_size, int pc_size, int alpha_size)
Oh! That function allocate the memory for the class matrix (array, that holds values of coefficients).
- void [Get](#) ([Grid](#) &grid, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
Function Get will get the class by loading it or calculating, whatever parameters-structure, loaded from the parameters-file tells it to do.
- bool [LoadDiffusionCoefficient](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, string D_filename, string filetype="IFT_GRID")
Loads diffusion coefficients Call other functions depends on filetype - file type.
- bool [LoadDiffusionCoefficientFromFileWithGrid](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, string D_filename, string gridOrder="IFT_GRID_LPA")
Load Dxx from the file with [Grid](#).
- bool [LoadDiffusionCoefficientFromPlaneFile](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, string D_filename)
Load Dxx from file without grid.
- void [Calculate](#) ([GridElement](#) &L, [GridElement](#) &epc, [GridElement](#) &alpha, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
function calculate Dxx
- [DiffusionCoefficient](#) & operator= (const [Matrix3D](#)< double > &M)
Operator Dxx = Matrix.
- [DiffusionCoefficient](#) & operator= (double val)
Operator Dxx = Value.
- [DiffusionCoefficient](#) operator* (double val)
*Operator Dxx*Value.*
- [DiffusionCoefficient](#) operator* ([Matrix3D](#)< double > &M)
*Operator Dxx*Matrix (not a matrix multiplication).*
- [DiffusionCoefficient](#) operator+ ([Matrix3D](#)< double > &M)
Operator Dxx + Matrix.
- void [MakeDLL](#) (double Kp)
Making DLL.
- void [MakeDLL](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double Kp, string DLLType="DLLT_B")
Making DLL.
- void [MakeDLL_B](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double Kp)
Making DLL.
- void [MakeDLL_FAKE](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double Kp)

Making DLL.

- void [MakeDLL_BE_res](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double Kp)

Making DLL.

- void [MakeDLL_BE](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double Kp)

Making DLL.

- void [MakeDLL100](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double Kp)

Making DLL.

Public Attributes

- [DiffusionCoefficientParamStructure Dxx_parameters](#)
- string [type](#)

Type of the diffusion coefficient: Daa, Dpp, Dpa etc... Described in types.h file as an enumeration.

- bool [is_active](#)

flag, if is enabled right now

- bool [useScale](#)

flag, if scale is appplayable

8.4.1 Detailed Description

Class [DiffusionCoefficient](#) holds diffusion coefficient matrix and routines to load and calculate it.

It gets [DiffusionCoefficientParamStructure](#) as an input with all waves parameters.

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

8.4.2 Constructor & Destructor Documentation

8.4.2.1 [DiffusionCoefficient::DiffusionCoefficient](#) (int *L_size*, int *pc_size*, int *alpha_size*) [inline]

Constructor. Also runs parent class constructor [Matrix3D<double>](#)(*L_size*, *pc_size*, *alpha_size*) and it's actually all it does.

Parameters:

L_size - L size

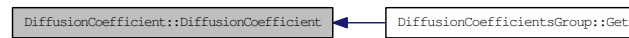
pc_size - pc size

alpha_size - alpha size

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

Referenced by [DiffusionCoefficientsGroup::Get\(\)](#).

Here is the caller graph for this function:



8.4.2.2 DiffusionCoefficient::DiffusionCoefficient (Grid & *grid*)

Constructor.

Constructur.

Parameters:

&grid - grid. Do you want me to name all classes as "..._class" and instances of classes as "..._instance" or capital and small letters are understandable enough?

Store grid inside the class, run constructor of parent class, mask initialization as false cause diffucion coefficient matrix was lon calculated/loaded.

Parameters:

Grid &grid - grid

Definition at line 50 of file DiffusionCoefficient.cpp.

References Matrix3D< double >::change_ind.

8.4.2.3 DiffusionCoefficient::DiffusionCoefficient (Grid & *grid*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Constructor.

Parameters:

&grid - grid? What to explain?

DxxParamStructure - structure with all diffusion coefficients parameters

Store grid inside the class, allocate memory for any arrays in class and get (calculate or load) diffusion coefficients

Parameters:

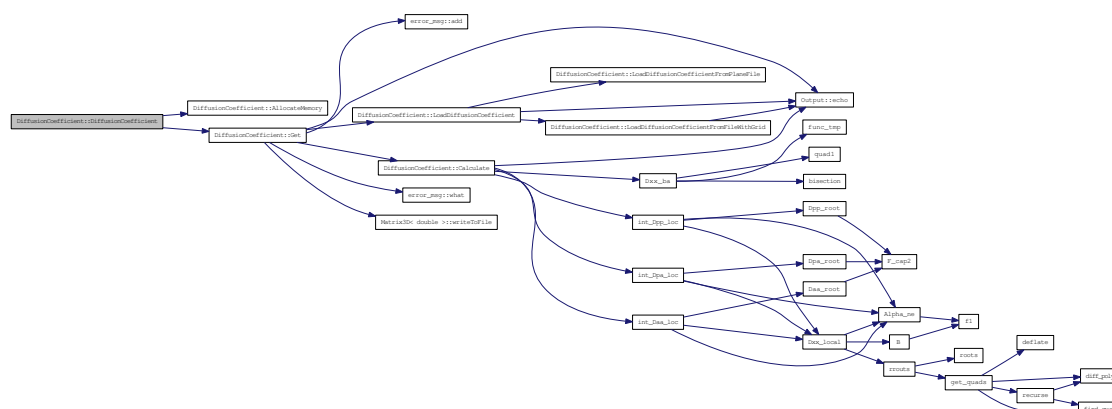
Grid &grid - grid

DiffusionCoefficientParamStructure DxxParamStructure - Structure with diffusion coefficients parameters

Definition at line 37 of file DiffusionCoefficient.cpp.

References AllocateMemory(), Grid::alpha, Matrix3D< double >::change_ind, Get(), Grid::L, Grid::pc, and GridElement::size.

Here is the call graph for this function:



8.4.3 Member Function Documentation

8.4.3.1 void DiffusionCoefficient::AllocateMemory (int *L_size*, int *pc_size*, int *alpha_size*)

Oh! That function allocate the memory for the class matrix (array, that holds values of coefficients).

Function allocate memory for parent matrix class by running allocating memory function of that parent matrix class.

Parameters:

L_size - L size

pc_size - pc size

alpha_size - alpha size

int *L_size* - L size

int *pc_size* - pc size

int *alpha_size* alpha size

Reimplemented from [Matrix3D< double >](#).

Definition at line 62 of file DiffusionCoefficient.cpp.

Referenced by DiffusionCoefficient().

Here is the caller graph for this function:



8.4.3.2 void DiffusionCoefficient::Calculate (GridElement & *L*, GridElement & *epc*, GridElement & *Alpha*, DiffusionCoefficientParamStructure *DxxParamStructure*)

function calculate Dxx

Calculating the diffusion coefficients.

Parameters:

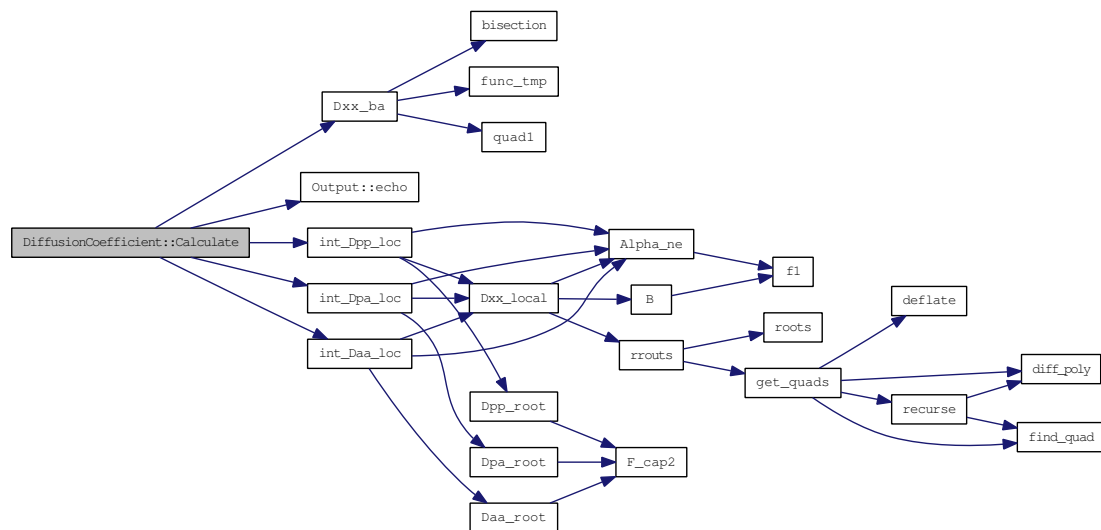
- GridElement** &L - **Grid** element L
- GridElement** &epc - **Grid** element epc
- GridElement** &Alpha - **Grid** element Alpha
- DiffusionCoefficientParamStructure** DxxParamStructure - Dxx parameters structure

Definition at line 136 of file DiffusionCoefficient.cpp.

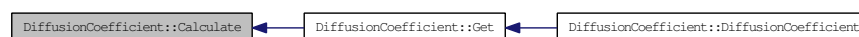
References Dxx_ba(), DiffusionCoefficientParamStructure::DxxName, DiffusionCoefficientParamStructure::DxxType, Output::echo(), int_Daa_loc(), int_Dpa_loc(), int_Dpp_loc(), min_Dxx, GridElement::size, and DiffusionCoefficientParamStructure::waveName.

Referenced by Get().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.3 void DiffusionCoefficient::Get (Grid &grid, DiffusionCoefficientParamStructure DxxParamStructure)

Function Get will get the class by loading it or calculating, whatever parameters-structure, loaded from the parameters-file tells it to do.

Function Get - return diffusion coefficient by loading or calculating it Two way of call function: With or without grid parameter If it called without grid parameter, it takes grid stored in the class : Remove grid from the class DiffusionCoefficients and remove functions, these assume it there.

Parameters:

&grid - a grid

DxxParamStructure - structure with diffusion coefficients parameters

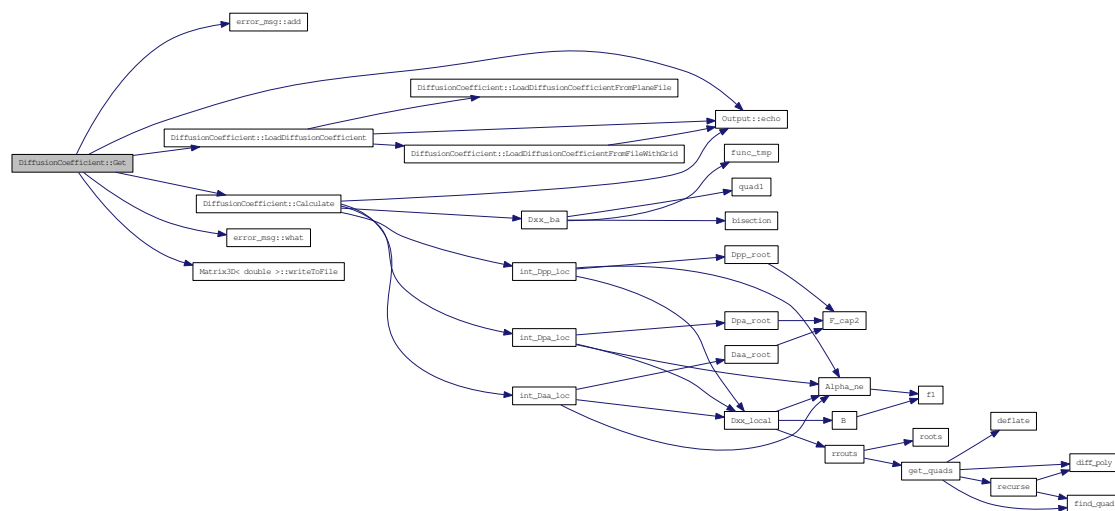
DiffusionCoefficientParamStructure DxxParamStructure - diffusion coefficients param structure

Definition at line 76 of file DiffusionCoefficient.cpp.

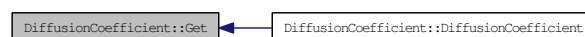
References `error_msg::add()`, `Grid::alpha`, `Calculate()`, `Dxx_parameters`, `DiffusionCoefficientParamStructure::DxxName`, `DiffusionCoefficientParamStructure::DxxType`, `Output::echo()`, `Grid::epc`, `err`, `DiffusionCoefficientParamStructure::filename`, `DiffusionCoefficientParamStructure::filetype`, `is_active`, `Grid::L`, `LoadDiffusionCoefficient()`, `DiffusionCoefficientParamStructure::loadOrCalculate`, `DiffusionCoefficientParamStructure::MLT_averaging`, `DiffusionCoefficientParamStructure::multiplier`, `Matrix3D< double >::name`, `Grid::pc`, `type`, `DiffusionCoefficientParamStructure::useScale`, `useScale`, `DiffusionCoefficientParamStructure::waveName`, `error_msg::what()`, and `Matrix3D< double >::writeToFile()`.

Referenced by `DiffusionCoefficient()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.4 bool DiffusionCoefficient::LoadDiffusionCoefficient (GridElement & L, GridElement & pc, GridElement & alpha, string D_filename, string filetype = "IFT_GRID")

Loads diffusion coefficients Call other functions depends on filetype - file type.

Loading coefficient from file runs loading function according file format (filetype).

Parameters:

&L - Grid element - L

&pc - Grid element - pc

&alpha - One more grid element - alpha

D_filename - filename to load diffusion coefficient from

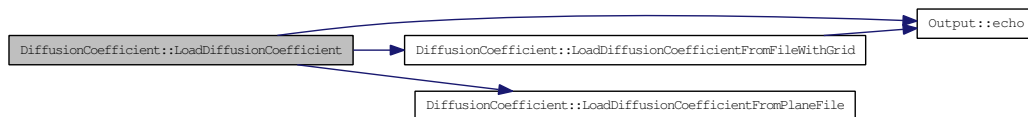
filetype = IFT_GRID grid type inside the file (yes, it can be different)

Definition at line 182 of file DiffusionCoefficient.cpp.

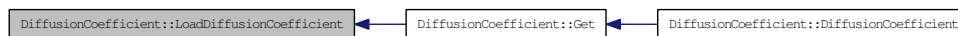
References Output::echo(), LoadDiffusionCoefficientFromFileWithGrid(), and LoadDiffusionCoefficientFromPlaneFile().

Referenced by Get().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.5 bool DiffusionCoefficient::LoadDiffusionCoefficientFromFileWithGrid (GridElement & L, GridElement & pc, GridElement & alpha, string D_filename, string gridOrder = "IFT_GRID_LPA")

Load Dxx from the file with [Grid](#).

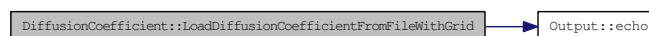
Loading coef from file with grid.

Definition at line 237 of file DiffusionCoefficient.cpp.

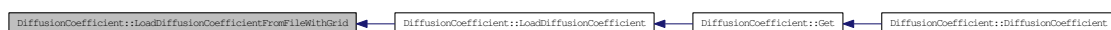
References Output::echo(), err, and GridElement::size.

Referenced by LoadDiffusionCoefficient().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.6 bool DiffusionCoefficient::LoadDiffusionCoefficientFromPlaneFile (GridElement & L, GridElement & pc, GridElement & alpha, string D_filename)

Load Dxx from file without grid.

Loading coefficient from plane file.

Definition at line 207 of file DiffusionCoefficient.cpp.

References GridElement::size.

Referenced by LoadDiffusionCoefficient().

Here is the caller graph for this function:



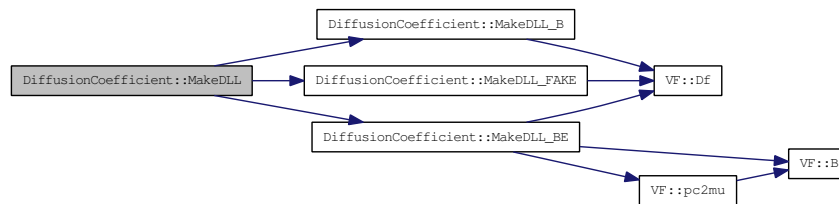
8.4.3.7 void DiffusionCoefficient::MakeDLL (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *Kp*, string *DLLType* = "DLLT_B")

Making DLL.

Definition at line 301 of file DiffusionCoefficient.cpp.

References MakeDLL_B(), MakeDLL_BE(), and MakeDLL_FAKE().

Here is the call graph for this function:

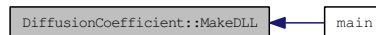


8.4.3.8 void DiffusionCoefficient::MakeDLL (double *Kp*)

Making DLL.

Referenced by main().

Here is the caller graph for this function:



8.4.3.9 void DiffusionCoefficient::MakeDLL100 (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *Kp*)

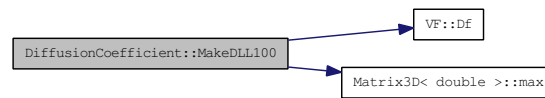
Making DLL.

Other version of DLL making procedure.

Definition at line 374 of file DiffusionCoefficient.cpp.

References VF::Df(), Matrix3D< double >::max(), and GridElement::size.

Here is the call graph for this function:



8.4.3.10 void DiffusionCoefficient::MakeDLL_B (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *Kp*)

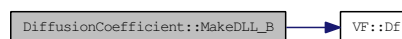
Making DLL.

Definition at line 317 of file DiffusionCoefficient.cpp.

References VF::Df(), and GridElement::size.

Referenced by MakeDLL().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.11 void DiffusionCoefficient::MakeDLL_BE (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *Kp*)

Making DLL.

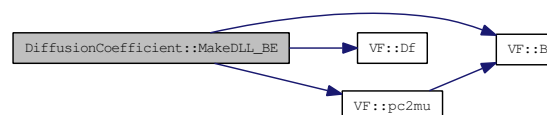
Other version of DLL making procedure.

Definition at line 343 of file DiffusionCoefficient.cpp.

References VF::B(), VF::Df(), VF::pc2mu(), and GridElement::size.

Referenced by MakeDLL().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.12 void DiffusionCoefficient::MakeDLL_BE_res (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *Kp*)

Making DLL.

8.4.3.13 void DiffusionCoefficient::MakeDLL_FAKE (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *Kp*)

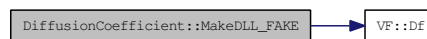
Making DLL.

Definition at line 330 of file DiffusionCoefficient.cpp.

References VF::Df(), and GridElement::size.

Referenced by MakeDLL().

Here is the call graph for this function:



Here is the caller graph for this function:



8.4.3.14 DiffusionCoefficient DiffusionCoefficient::operator* (Matrix3D< double > & *M*)

Operator Dxx*Matrix (not a matrix multiplication).

Operator Dxx * Matrix (not a matrix operation).

Reimplemented from [Matrix3D< double >](#).

Definition at line 419 of file DiffusionCoefficient.cpp.

References [Matrix3D< T >::Matrix3D\(\)](#).

Here is the call graph for this function:



8.4.3.15 DiffusionCoefficient DiffusionCoefficient::operator* (double *val*)

Operator Dxx*Value.

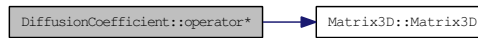
Operator Dxx * value.

Reimplemented from [Matrix3D< double >](#).

Definition at line 403 of file DiffusionCoefficient.cpp.

References [Matrix3D< T >::Matrix3D\(\)](#).

Here is the call graph for this function:



8.4.3.16 DiffusionCoefficient DiffusionCoefficient::operator+ (Matrix3D< double > &M)

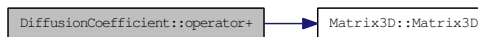
Operator Dxx + Matrix.

Reimplemented from [Matrix3D< double >](#).

Definition at line 411 of file DiffusionCoefficient.cpp.

References [Matrix3D< T >::Matrix3D\(\)](#).

Here is the call graph for this function:



8.4.3.17 DiffusionCoefficient & DiffusionCoefficient::operator= (double val)

Operator Dxx = Value.

Reimplemented from [Matrix3D< double >](#).

Reimplemented in [DiffusionCoefficientsGroup](#).

Definition at line 396 of file DiffusionCoefficient.cpp.

References [operator=\(\)](#).

Here is the call graph for this function:



8.4.3.18 DiffusionCoefficient & DiffusionCoefficient::operator= (const Matrix3D< double > &M)

Operator Dxx = Matrix.

Reimplemented from [Matrix3D< double >](#).

Reimplemented in [DiffusionCoefficientsGroup](#).

Definition at line 389 of file DiffusionCoefficient.cpp.

Referenced by [operator=\(\)](#).

Here is the caller graph for this function:



8.4.3.19 double DiffusionCoefficient::Scale (double *Kp*)

Function, that scale diffusion coefficients.

Scaling procedure.

Todo

Upgrade scaling diffusion coefficients procedure with scaling according to external array

Definition at line 558 of file DiffusionCoefficient.cpp.

References `Dxx_parameters`, `DiffusionCoefficientParamStructure::DxxKp`, `useScale`, and `DiffusionCoefficientParamStructure::waveType`.

8.4.4 Member Data Documentation

8.4.4.1 DiffusionCoefficientParamStructure DiffusionCoefficient::Dxx_parameters

Definition at line 27 of file DiffusionCoefficient.h.

Referenced by `DiffusionCoefficientsGroup::ActivateAndScale()`, `Get()`, and `Scale()`.

8.4.4.2 bool DiffusionCoefficient::is_active

flag, if is enabled right now

Definition at line 35 of file DiffusionCoefficient.h.

Referenced by `DiffusionCoefficientsGroup::ActivateAndScale()`, and `Get()`.

8.4.4.3 string DiffusionCoefficient::type

Type of the diffusion coefficient: Daa, Dpp, Dpa etc... Described in `types.h` file as an enumeration.

Definition at line 32 of file DiffusionCoefficient.h.

Referenced by `DiffusionCoefficientsGroup::Get()`, and `Get()`.

8.4.4.4 bool DiffusionCoefficient::useScale

flag, if scale is appplayable

Definition at line 38 of file DiffusionCoefficient.h.

Referenced by `DiffusionCoefficientsGroup::ActivateAndScale()`, `Get()`, and `Scale()`.

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

- [DiffusionCoefficient.h](#)
- [DiffusionCoefficient.cpp](#)

8.5 DiffusionCoefficientParamStructure Struct Reference

Diffusion coefficient parameters.

```
#include <Parameters.h>
```

Public Member Functions

- bool [Load_dxx_parameters](#) (string [filename](#))
Loads parameters from file.

Public Attributes

- string [DxxType](#)
Dxx type. Check StrToVal(string input, DiffusionCoefficientTypes &place) for known values.
- string [DxxName](#)
Dxx name.
- string [waveType](#)
Wave type (chorus, hiss etc). Check StrToVal(string input, WaveTypes &place) for known values.
- string [waveName](#)
Wave name.
- string [filetype](#)
Type of the file (w/wo grid etc).
- string [filename](#)
File name for loading ot saving diffusion coefficients array.
- double [time_start](#)
Time, when diffusion coefficient starts.
- double [time_end](#)
Time, when diffusion coefficient ends.
- bool [useScale](#)
Flag, use scaling if equal to true.
- double [DxxKp](#)
Kp value associated with parameters (for Kp-scaling).
- string [loadOrCalculate](#)
Load diffusion coefficient or calculate flag.
- string [numberDensity](#)
Number density model. Check StrToVal(string input, NumberDencities &place) for known values.

- double [MLT_averaging](#)
MLT averaging part of the orbit.
- double [multiplier](#)
Dxx is multiplied to this number after all other operations. Useful for fast Dxx modifications.
- string [Omega_mType](#)
Omega_m type. Check StrToVal(string input, Omega_mTypes &place) for known values.
- double [Omega_m](#)
Omega_m value.
- double [d_omega](#)
d_omega value.
- double [omega_uc](#)
omega upper cutoff
- double [omega_lc](#)
omega lower cutoff
- double [eta1](#)
- double [eta2](#)
- double [eta3](#)
- double [Bw](#)
Bw value.
- bool [BwFromLambda](#)
Bw lambda dependance flag.
- int [nint](#)
- double [lam_min](#)
Minimum latitude.
- double [lam_max](#)
Maximum latitude.
- double [nu](#)
- double [s](#)
- double [f](#)
- string [particle](#)
Type of particles, produced wave? Ions or electrons. Check StrToVal(string input, ParticleTypes &place) for known values.
- double [L](#)
- double [EMeV](#)
- double [Alpha](#)

8.5.1 Detailed Description

Diffusion coefficient parameters.

Definition at line 25 of file Parameters.h.

8.5.2 Member Function Documentation

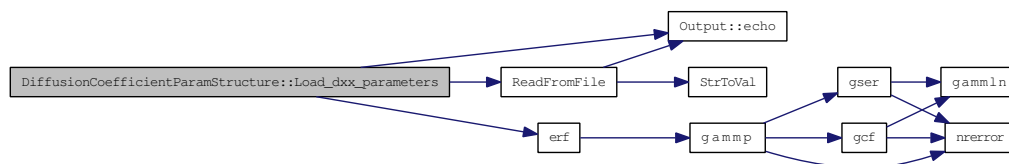
8.5.2.1 `bool DiffusionCoefficientParamStructure::Load_dxx_parameters (string filename)`

Loads parameters from file.

Definition at line 408 of file Parameters.cpp.

References Bw, BwFromLambda, d_omega, DxxKp, DxxName, DxxType, Output::echo(), erf(), eta1, eta2, eta3, f, filename, filetype, lam_max, lam_min, loadOrCalculate, MLT_averaging, multiplier, nint, nu, numberDensity, omega_lc, Omega_m, Omega_mType, omega_uc, particle, ReadFromFile(), s, time_end, time_start, useScale, waveName, and waveType.

Here is the call graph for this function:



8.5.3 Member Data Documentation

8.5.3.1 `double DiffusionCoefficientParamStructure::Alpha`

Definition at line 71 of file Parameters.h.

Referenced by `Dxx_ba()`, `Dxx_local()`, `int_Daa_loc()`, `int_Dpa_loc()`, and `int_Dpp_loc()`.

8.5.3.2 `double DiffusionCoefficientParamStructure::Bw`

Bw value.

Definition at line 57 of file Parameters.h.

Referenced by `Dxx_local()`, and `Load_dxx_parameters()`.

8.5.3.3 `bool DiffusionCoefficientParamStructure::BwFromLambda`

Bw lambda dependance flag.

Definition at line 58 of file Parameters.h.

Referenced by `Dxx_local()`, and `Load_dxx_parameters()`.

8.5.3.4 double DiffusionCoefficientParamStructure::d_omega

d_omega value.

Definition at line 51 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.5 double DiffusionCoefficientParamStructure::DxxKp

Kp value associated with parameters (for Kp-scaling).

Definition at line 40 of file Parameters.h.

Referenced by Load_dxx_parameters(), and DiffusionCoefficient::Scale().

8.5.3.6 string DiffusionCoefficientParamStructure::DxxName

Dxx name.

Definition at line 28 of file Parameters.h.

Referenced by DiffusionCoefficient::Calculate(), DiffusionCoefficient::Get(), and Load_dxx_parameters().

8.5.3.7 string DiffusionCoefficientParamStructure::DxxType

Dxx type. Check StrToVal(string input, DiffusionCoefficientTypes &place) for known values.

Definition at line 27 of file Parameters.h.

Referenced by DiffusionCoefficient::Calculate(), DiffusionCoefficient::Get(), and Load_dxx_parameters().

8.5.3.8 double DiffusionCoefficientParamStructure::EMeV

Definition at line 71 of file Parameters.h.

Referenced by Daa_root(), Dpa_root(), Dpp_root(), Dxx_ba(), and Dxx_local().

8.5.3.9 double DiffusionCoefficientParamStructure::eta1

Definition at line 55 of file Parameters.h.

Referenced by Dxx_local(), F_cap2(), and Load_dxx_parameters().

8.5.3.10 double DiffusionCoefficientParamStructure::eta2

Definition at line 55 of file Parameters.h.

Referenced by Dxx_local(), F_cap2(), and Load_dxx_parameters().

8.5.3.11 double DiffusionCoefficientParamStructure::eta3

Definition at line 55 of file Parameters.h.

Referenced by `Dxx_local()`, `F_cap2()`, and `Load_dxx_parameters()`.

8.5.3.12 double DiffusionCoefficientParamStructure::f

Definition at line 65 of file `Parameters.h`.

Referenced by `Dxx_local()`, and `Load_dxx_parameters()`.

8.5.3.13 string DiffusionCoefficientParamStructure::filename

File name for loading or saving diffusion coefficients array.

Definition at line 34 of file `Parameters.h`.

Referenced by `DiffusionCoefficient::Get()`, and `Load_dxx_parameters()`.

8.5.3.14 string DiffusionCoefficientParamStructure::filetype

Type of the file (w/wo grid etc).

Definition at line 33 of file `Parameters.h`.

Referenced by `DiffusionCoefficient::Get()`, and `Load_dxx_parameters()`.

8.5.3.15 double DiffusionCoefficientParamStructure::L

Definition at line 71 of file `Parameters.h`.

Referenced by `Dxx_ba()`, `Dxx_local()`, `int_Daa_loc()`, `int_Dpa_loc()`, and `int_Dpp_loc()`.

8.5.3.16 double DiffusionCoefficientParamStructure::lam_max

Maximum latitude.

Definition at line 61 of file `Parameters.h`.

Referenced by `Dxx_ba()`, and `Load_dxx_parameters()`.

8.5.3.17 double DiffusionCoefficientParamStructure::lam_min

Minimum latitude.

Definition at line 60 of file `Parameters.h`.

Referenced by `Dxx_ba()`, and `Load_dxx_parameters()`.

8.5.3.18 string DiffusionCoefficientParamStructure::loadOrCalculate

Load diffusion coefficient or calculate flag.

Definition at line 42 of file `Parameters.h`.

Referenced by `DiffusionCoefficient::Get()`, and `Load_dxx_parameters()`.

8.5.3.19 double DiffusionCoefficientParamStructure::MLT_averaging

MLT averaging part of the orbit.

Definition at line 45 of file Parameters.h.

Referenced by DiffusionCoefficient::Get(), and Load_dxx_parameters().

8.5.3.20 double DiffusionCoefficientParamStructure::multiplier

Dxx is multiplied to this number after all other operations. Useful for fast Dxx modifications.

Definition at line 47 of file Parameters.h.

Referenced by DiffusionCoefficient::Get(), and Load_dxx_parameters().

8.5.3.21 int DiffusionCoefficientParamStructure::nint

Definition at line 59 of file Parameters.h.

Referenced by Dxx_ba(), and Load_dxx_parameters().

8.5.3.22 double DiffusionCoefficientParamStructure::nu

Definition at line 63 of file Parameters.h.

Referenced by Daa_root(), Dpa_root(), Dpp_root(), and Load_dxx_parameters().

8.5.3.23 string DiffusionCoefficientParamStructure::numberDensity

Number density model. Check StrToVal(string input, NumberDencities &place) for known values.

Definition at line 44 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.24 double DiffusionCoefficientParamStructure::omega_lc

omega lower cutoff

Definition at line 53 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.25 double DiffusionCoefficientParamStructure::Omega_m

Omega_m value.

Definition at line 50 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.26 string DiffusionCoefficientParamStructure::Omega_mType

Omega_m type. Check StrToVal(string input, Omega_mTypes &place) for known values.

Definition at line 49 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.27 double DiffusionCoefficientParamStructure::omega_uc

omega upper cutoff

Definition at line 52 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.28 string DiffusionCoefficientParamStructure::particle

Type of particles, produced wave? Ions or electrons. Check StrToVal(string input, ParticleTypes &place) for known values.

Definition at line 68 of file Parameters.h.

Referenced by Dxx_local(), and Load_dxx_parameters().

8.5.3.29 double DiffusionCoefficientParamStructure::s

Definition at line 64 of file Parameters.h.

Referenced by Dxx_local(), Load_dxx_parameters(), and rrouts().

8.5.3.30 double DiffusionCoefficientParamStructure::time_end

Time, when diffusion coefficient ends.

Definition at line 37 of file Parameters.h.

Referenced by Load_dxx_parameters().

8.5.3.31 double DiffusionCoefficientParamStructure::time_start

Time, when diffusion coefficient starts.

Definition at line 36 of file Parameters.h.

Referenced by DiffusionCoefficientsGroup::ActivateAndScale(), and Load_dxx_parameters().

8.5.3.32 bool DiffusionCoefficientParamStructure::useScale

Flag, use scaling if equal to true.

Definition at line 39 of file Parameters.h.

Referenced by DiffusionCoefficient::Get(), and Load_dxx_parameters().

8.5.3.33 string DiffusionCoefficientParamStructure::waveName

Wave name.

Definition at line 31 of file Parameters.h.

Referenced by DiffusionCoefficient::Calculate(), DiffusionCoefficient::Get(), and Load_dxx_parameters().

8.5.3.34 string DiffusionCoefficientParamStructure::waveType

Wave type (chorus, hiss etc). Check StrToVal(string input, WaveTypes &place) for known values.

Definition at line 30 of file Parameters.h.

Referenced by Load_dxx_parameters(), and DiffusionCoefficient::Scale().

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

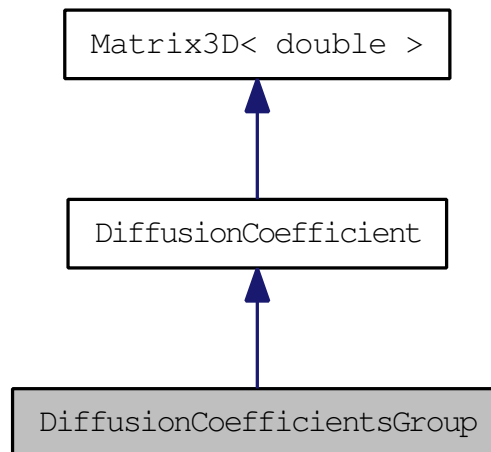
- [Parameters.h](#)
- [Parameters.cpp](#)

8.6 DiffusionCoefficientsGroup Class Reference

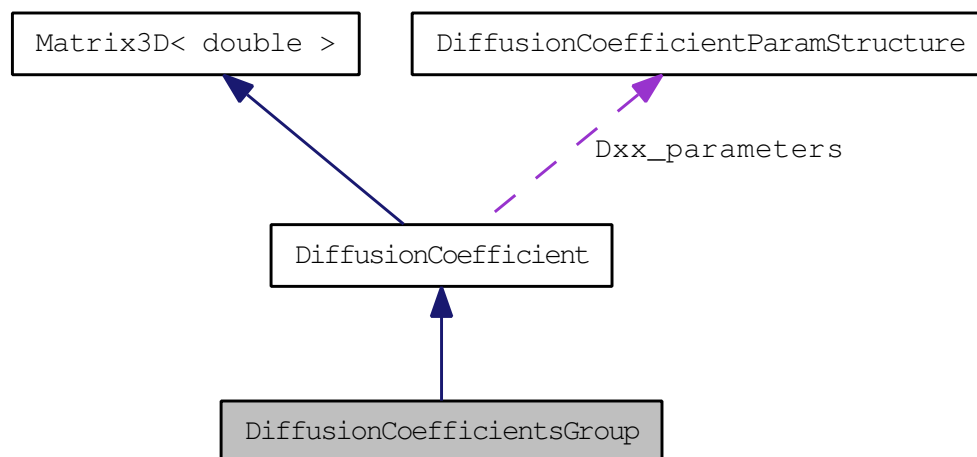
It has [DiffusionCoefficient](#) class as a parent class, so it stores there summation of all the coefficients to use in diffusion.

```
#include <DiffusionCoefficient.h>
```

Inheritance diagram for DiffusionCoefficientsGroup:



Collaboration diagram for DiffusionCoefficientsGroup:



Public Member Functions

- bool [ActivateAndScale](#) (double time, double Kp)

Function activate (enable/disable) diffusion coefficients according time events and scale diffusion coefficients. It would be cool to call it "Actualize", but unfortunately it would be too complicated.

- [DiffusionCoefficientsGroup](#) (string type, [Grid](#) &grid)

Constructor: It save passed in type and grid to this->type and this->grid. "this → " in C++ means... whatever, it means local.

- [DiffusionCoefficientsGroup](#) (string [type](#), int L_size, int pc_size, int alpha_size)
Constructor: That constructor save type and runs constructor of the parent class - [DiffusionCoefficient](#).
- [DiffusionCoefficientsGroup](#) (string [type](#), [Grid](#) &grid, [DiffusionCoefficientParamStructureList](#) DxxParamStructureList)
Constructor: Save the type, the grid and get (load or calculate) all diffusion coefficients with the type from the DxxParamStructureList.
- void [Get](#) ([DiffusionCoefficientParamStructureList](#) DxxParamStructureList)
Get the diffusion coefficients matrix values (load or calculate).
- void [Get](#) ([DiffusionCoefficientParamStructureList](#) DxxParamStructureList, string [type](#))
Get the diffusion coefficients matrix values (load or calculate).
- void [Get](#) ([Grid](#) &grid, [DiffusionCoefficientParamStructureList](#) DxxParamStructureList)
Get the diffusion coefficients matrix values (load or calculate).
- void [Get](#) ([Grid](#) &grid, [DiffusionCoefficientParamStructureList](#) DxxParamStructureList, string [type](#))
Get the diffusion coefficients matrix values (load or calculate).
- [DiffusionCoefficientsGroup](#) & [operator=](#) (double val)
Operator = val.
- [DiffusionCoefficientsGroup](#) & [operator=](#) (const [Matrix3D](#)< double > &M)
Operator = Matrix.
- [DiffusionCoefficientsGroup](#) & [operator=](#) (const [DiffusionCoefficient](#) &Dxx)
Operator = Dxx.

Public Attributes

- vector< [DiffusionCoefficient](#) > [DxxList](#)
List of diffusion coefficients in that group. Actually, it's a list of waves used in the diffusion coefficient that this group represent.

8.6.1 Detailed Description

It has [DiffusionCoefficient](#) class as a parent class, so it stores there summation of all the coefficients to use in diffusion.

So, it is [DiffusionCoefficient](#) in common cense.

Todo

change [DiffusionCoefficientGroup](#) name to the [DiffusionCoefficient](#) name and [DiffusionCoefficient](#) name to something else.

Holds list of instances of [DiffusionCoefficient](#) class of same type (like Daa, Dpp, etc), but produced by different waves (Daa_chorus, Daa_EMITC, etc).

Definition at line 131 of file DiffusionCoefficient.h.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 DiffusionCoefficientsGroup::DiffusionCoefficientsGroup (string *type*, Grid & *grid*)

Constructor. It save passed in type and grid to this->type and this->grid. "this → " in C++ means... whatever, it means local.

[DiffusionCoefficientsGroup](#) constructor Didn't I mention, that the name of the class should be changed?

Definition at line 430 of file DiffusionCoefficient.cpp.

8.6.2.2 DiffusionCoefficientsGroup::DiffusionCoefficientsGroup (string *type*, int *L_size*, int *pc_size*, int *alpha_size*)

Constructor. That constructor save type and runs constructor of the parent class - [DiffusionCoefficient](#).

[DiffusionCoefficientsGroup](#) constructor.

Parameters:

type - Type. Of the parent diffusion coefficient.

L_size - L size

pc_size - pc size

alpha_size - alpha size

Definition at line 439 of file DiffusionCoefficient.cpp.

8.6.2.3 DiffusionCoefficientsGroup::DiffusionCoefficientsGroup (string *type*, Grid & *grid*, DiffusionCoefficientParamStructureList *DxxParamStructureList*)

Constructor. Save the type, the grid and get (load or calculate) all diffusion coefficients with the type from the DxxParamStructureList.

[DiffusionCoefficientsGroup](#) constructor.

Parameters:

type - diffusion coefficient type.

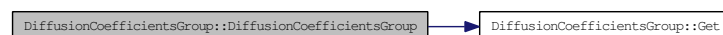
&grid - grid

DxxParamStructureList - damn, that's the list (vector) of diffusion coefficients param structure. What the point to duble all variables names???

Definition at line 448 of file DiffusionCoefficient.cpp.

References Get().

Here is the call graph for this function:



8.6.3 Member Function Documentation

8.6.3.1 bool DiffusionCoefficientsGroup::ActivateAndScale (double *time*, double *Kp*)

Function activate (enable/disable) diffusion coefficients according time events and scale diffusion coefficients. It would be cool to call it "Actualize", but unfortunately it would be too complicated.

Activation (enabling/disabling) and scaling of the diffusion coefficients in one group.

Parameters:

time - time

Kp - Kp

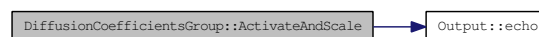
It loops thru the list of the diffusion coefficients, enable which needs to be enabled, disable, which needs to be disabled and scale all enabled

Definition at line 493 of file DiffusionCoefficient.cpp.

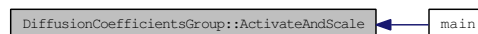
References `Matrix3D< double >::change_ind`, `DiffusionCoefficient::Dxx_parameters`, `DxxList`, `Output::echo()`, `DiffusionCoefficient::is_active`, `Matrix3D< double >::name`, `DiffusionCoefficientParamStructure::time_start`, and `DiffusionCoefficient::useScale`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.6.3.2 void DiffusionCoefficientsGroup::Get (Grid & *grid*, DiffusionCoefficientParamStructureList *DxxParamStructureList*, string *type*)

Get the diffusion coefficients matrix values (load or calculate).

Calculating all diffusion coefficients in the class.

Type and grid are taken from the parameters. It makes list of the diffusion coefficients with the same type (Daa_chorus, Daa_hiss etc). Each time new diffusion coefficient is pushed in the list, it's constructor is runned and construct it. Isn't easier just to look to the 2 lines of the code, then read all these multiple lines of unclear comments?

Definition at line 472 of file DiffusionCoefficient.cpp.

References `Matrix3D< double >::change_ind`, `DiffusionCoefficient::DiffusionCoefficient()`, `DxxList`, and `Matrix3D< double >::name`.

Here is the call graph for this function:



8.6.3.3 void DiffusionCoefficientsGroup::Get (Grid & grid, DiffusionCoefficientParamStructureList DxxParamStructureList)

Get the diffusion coefficients matrix values (load or calculate).

Calculation of all diffusion coefficients in the group Taking grid from the parameters and type from the class And pass everything to the calculating function.

Definition at line 461 of file DiffusionCoefficient.cpp.

References Get(), and DiffusionCoefficient::type.

Here is the call graph for this function:



8.6.3.4 void DiffusionCoefficientsGroup::Get (DiffusionCoefficientParamStructureList DxxParamStructureList, string type)

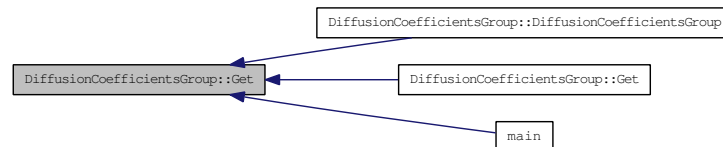
Get the diffusion coefficients matrix values (load or calculate).

8.6.3.5 void DiffusionCoefficientsGroup::Get (DiffusionCoefficientParamStructureList DxxParamStructureList)

Get the diffusion coefficients matrix values (load or calculate).

Referenced by DiffusionCoefficientsGroup(), Get(), and main().

Here is the caller graph for this function:



8.6.3.6 DiffusionCoefficientsGroup & DiffusionCoefficientsGroup::operator= (const DiffusionCoefficient & Dxx)

Operator = Dxx.

operator DxxGroup = val Makes Dxx = Matrix for parent class

Definition at line 600 of file DiffusionCoefficient.cpp.

References operator=().

Here is the call graph for this function:



8.6.3.7 DiffusionCoefficientsGroup & DiffusionCoefficientsGroup::operator= (const Matrix3D<double> & M)

Operator = Matrix.

operator DxxGroup = val Makes Dxx = Matrix for parent class

Reimplemented from [DiffusionCoefficient](#).

Definition at line 592 of file DiffusionCoefficient.cpp.

References operator=().

Here is the call graph for this function:



8.6.3.8 DiffusionCoefficientsGroup & DiffusionCoefficientsGroup::operator= (double val)

Operator = val.

operator DxxGroup = val Makes Dxx = val for parent class

Reimplemented from [DiffusionCoefficient](#).

Definition at line 583 of file DiffusionCoefficient.cpp.

Referenced by operator=().

Here is the caller graph for this function:



8.6.4 Member Data Documentation

8.6.4.1 vector<DiffusionCoefficient> DiffusionCoefficientsGroup::DxxList

List of diffusion coefficients in that group. Actually, it's a list of waves used in the diffusion coefficient that this group represent.

Definition at line 139 of file DiffusionCoefficient.h.

Referenced by ActivateAndScale(), Get(), and main().

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

- [DiffusionCoefficient.h](#)
- [DiffusionCoefficient.cpp](#)

8.7 error_msg Class Reference

Error message - stack of [single_error](#) s.

```
#include <error.h>
```

Public Member Functions

- [error_msg](#) ()
Constructor.
- [error_msg](#) (char *code)
Constructor.
- [error_msg](#) (const char *code, const char *msg,...)
Constructor.
- void [add](#) ([single_error](#) err)
Add one error message to the stack.
- void [add](#) (string code)
Add one error message to the stack.
- void [add](#) (string code, string msg)
Add one error message to the stack.
- string [what](#) ()
Return all messages as a text.

Public Attributes

- vector< [single_error](#) > [errors_stack](#)
Stack of errors.

8.7.1 Detailed Description

Error message - stack of [single_error](#) s.

Definition at line 54 of file error.h.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 error_msg::error_msg ()

Constructor.

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

Here is the caller graph for this function:



8.7.2.2 error_msg::error_msg(char *code) [inline]

Constructor.

one message added to the stack.

Definition at line 64 of file error.h.

References errors_stack.

8.7.2.3 error_msg::error_msg(const char *code, const char *msg, ...) [inline]

Constructor.

one message added to the stack.

```
!! len = _vscprintf( msg, args ) + 1;
```

Definition at line 75 of file error.h.

References error_msg(), and errors_stack.

Here is the call graph for this function:



8.7.3 Member Function Documentation

8.7.3.1 void error_msg::add(string code, string msg) [inline]

Add one error message to the stack.

Definition at line 110 of file error.h.

References errors_stack.

8.7.3.2 void error_msg::add(string code) [inline]

Add one error message to the stack.

Definition at line 103 of file error.h.

References errors_stack.

8.7.3.3 void error_msg::add(single_error err) [inline]

Add one error message to the stack.

Definition at line 96 of file error.h.

References errors_stack.

Referenced by DiffusionCoefficient::Get().

Here is the caller graph for this function:



8.7.3.4 string error_msg::what () [inline]

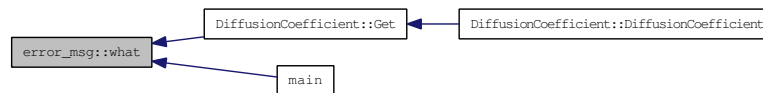
Return all messages as a text.

Definition at line 117 of file error.h.

References errors_stack, and i.

Referenced by DiffusionCoefficient::Get(), and main().

Here is the caller graph for this function:



8.7.4 Member Data Documentation

8.7.4.1 vector<single_error> error_msg::errors_stack

Stack of errors.

Definition at line 57 of file error.h.

Referenced by add(), error_msg(), and what().

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

- [error.h](#)

8.8 ParamStructure::General_Output_parameters Struct Reference

General programm output parameters structure.

```
#include <Parameters.h>
```

Public Attributes

- double [timeStep](#)
Time step for output.
- int [iterStep](#)
Number of iterations between outputs.
- string [logFileName](#)
log-file name.
- string [folderName](#)
output folder name.
- string [fileName1D](#)
1d-output file name.

8.8.1 Detailed Description

General programm output parameters structure.

Definition at line 116 of file Parameters.h.

8.8.2 Member Data Documentation

8.8.2.1 string ParamStructure::General_Output_parameters::fileName1D

1d-output file name.

Definition at line 121 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and main().

8.8.2.2 string ParamStructure::General_Output_parameters::folderName

output folder name.

Definition at line 120 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and main().

8.8.2.3 int ParamStructure::General_Output_parameters::iterStep

Number of iterations between outputs.

Definition at line 118 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and main().

8.8.2.4 string ParamStructure::General_Output_parameters::logFileName

log-file name.

Definition at line 119 of file Parameters.h.

Referenced by ParamStructure::Load_parameters().

8.8.2.5 double ParamStructure::General_Output_parameters::timeStep

Time step for output.

Definition at line 117 of file Parameters.h.

Referenced by ParamStructure::Load_parameters().

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

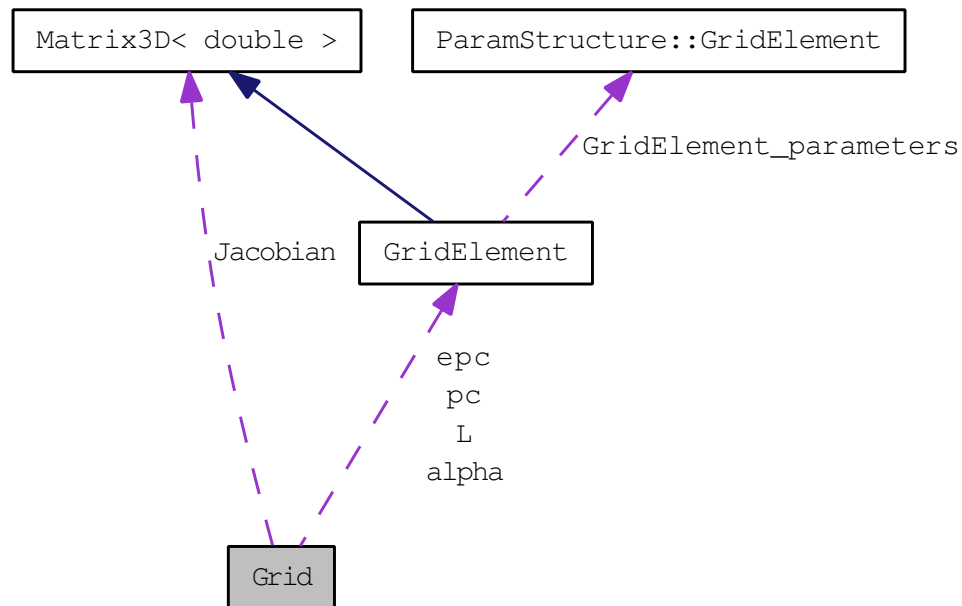
- [Parameters.h](#)

8.9 Grid Class Reference

Computational grid Combined from 3 grid elements: L, pc, Alpha and additional array of epc values for convenience.

```
#include <Grid.h>
```

Collaboration diagram for Grid:



Public Member Functions

- `Grid ()`
- `Grid (ParamStructure::GridElement parameters_L, ParamStructure::GridElement parameters_pc, ParamStructure::GridElement parameters_alpha, ParamStructure::GridElement parameters_epc, string grid_filename, string grid_type, Grid SecondGrid=Grid())`

Constructor.

- `void Initialize (ParamStructure::GridElement parameters_L, ParamStructure::GridElement parameters_pc, ParamStructure::GridElement parameters_alpha, ParamStructure::GridElement parameters_epc, string grid_type)`

Grid initialization.

- `void MakeGrid (string grid_type, Grid SecondGrid=Grid())`

Makes grid.

- `bool IsInitialized ()`
- `void Output (string filename)`

Making boundary conditions.

Public Attributes

- [GridElement L](#)
- [GridElement pc](#)
- [GridElement alpha](#)
- [GridElement epc](#)
- string [type](#)
- int [LSize](#)
- int [pcSize](#)
- int [alphaSize](#)
- int [epcSize](#)
- [Matrix3D< double > Jacobian](#)

Jacobian of the diffusion equation.

8.9.1 Detailed Description

Computational grid Combined from 3 grid elements: L, pc, Alpha and additional array of epc values for convenience.

Definition at line 65 of file Grid.h.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 `Grid::Grid ()` `[inline]`

Definition at line 75 of file Grid.h.

8.9.2.2 `Grid::Grid (ParamStructure::GridElement parameters_L, ParamStructure::GridElement parameters_pc, ParamStructure::GridElement parameters_alpha, ParamStructure::GridElement parameters_epc, string grid_filename, string grid_type, Grid SecondGrid = Grid ())`

Constructor.

Create grid that can be based on second grid (ortogonal local grid for radial grid)

Parameters:

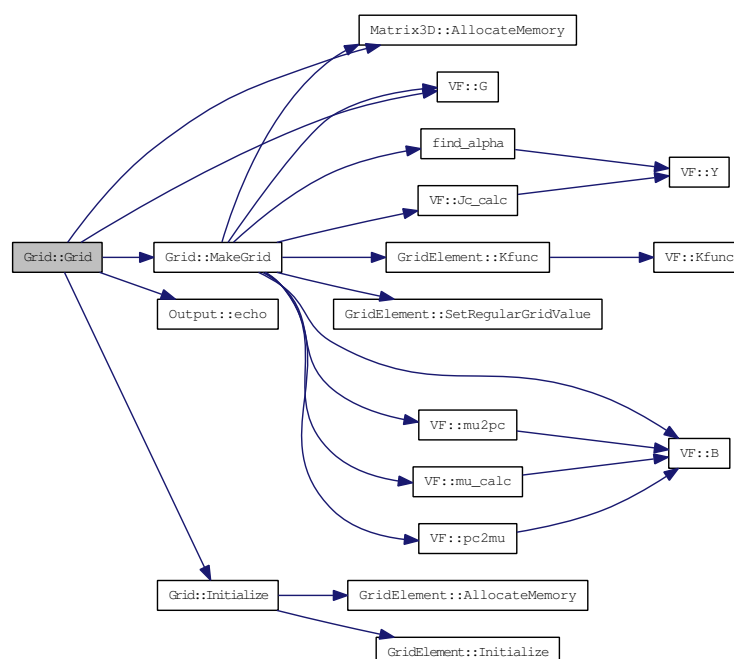
ParamStructure::Grid parameters - grid parameters structure

Grid SecondGrid - second grid (optional)

Definition at line 186 of file Grid.cpp.

References `Matrix3D< T >::AllocateMemory()`, `alpha`, `Output::echo()`, `epc`, `VF::G()`, `Initialize()`, `Jacobian`, `L`, `MakeGrid()`, `pc`, and `GridElement::size`.

Here is the call graph for this function:



8.9.3 Member Function Documentation

8.9.3.1 void Grid::Initialize (ParamStructure::GridElement *parameters_L*, ParamStructure::GridElement *parameters_pc*, ParamStructure::GridElement *parameters_alpha*, ParamStructure::GridElement *parameters_epc*, string *grid_type*)

[Grid](#) initialization.

Allocating memory for all gred alaments, copying parameters from paramStructure to the class.

Parameters:

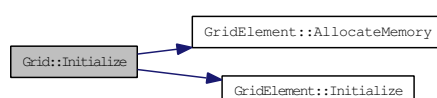
ParamStructure::Grid parameters - grid parameters structure

Definition at line 260 of file Grid.cpp.

References GridElement::AllocateMemory(), alpha, alphaSize, epc, epcSize, GridElement::Initialize(), L, LSize, pc, pcSize, GridElement::size, ParamStructure::GridElement::size, and type.

Referenced by Grid().

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.3.2 bool Grid::IsInitialized () [inline]

Definition at line 92 of file Grid.h.

8.9.3.3 void Grid::MakeGrid (string *gridType*, Grid *SecondGrid* = Grid ())

Makes grid.

Making grid routine accordint to grid type.

Parameters:

GridTypes *gridType* - grid type

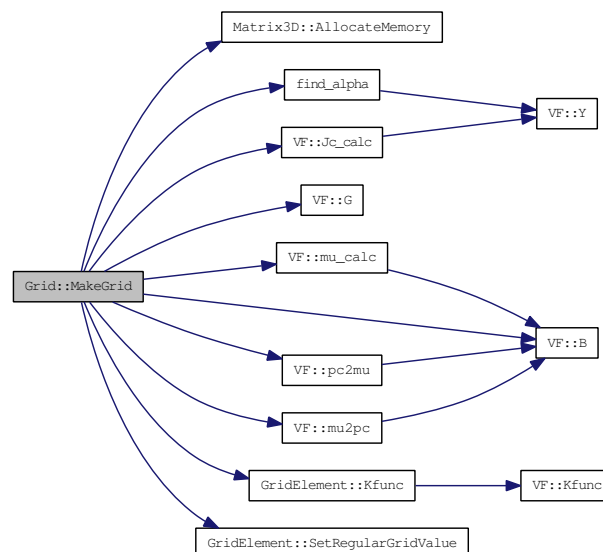
Grid *SecondGrid* - second grid (optional, depends on type).

Definition at line 291 of file Grid.cpp.

References Matrix3D< T >::AllocateMemory(), alpha, VF::B(), epc, find_alpha(), VF::G(), GridElement::GridElement_parameters, Jacobian, VF::Jc_calc(), GridElement::Kfunc(), L, ParamStructure::GridElement::max, maxERR, ParamStructure::GridElement::min, VF::mu2pc(), VF::mu_calc(), pc, VF::pc2mu(), GridElement::SetRegularGridValue(), GridElement::size, and ParamStructure::GridElement::useLogScale.

Referenced by Grid().

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.3.4 void Grid::Output (string filename)

Making boundary conditions.

Here we run MakeBoundaryCondition function for each boundary and pass there corresponding slice of [PSD](#) (just in case). If type of the boundary condition is constant [PSD](#), then passed array is used. In other cases (other types of boundary conditions) passed array is just ignored and boundary condition values are calculated according to the type (constant value or constant derivative).

Parameters:

[Matrix3D<double>](#) &psd - phase space density values array. [Output](#) grid to a file.

[string](#) filename - file name

Definition at line 512 of file Grid.cpp.

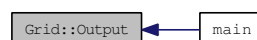
References [alpha](#), [epc](#), [L](#), [VF::pfunc\(\)](#), and [GridElement::size](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.9.4 Member Data Documentation

8.9.4.1 GridElement Grid::alpha

Definition at line 70 of file Grid.h.

Referenced by [DiffusionCoefficient::DiffusionCoefficient\(\)](#), [DiffusionCoefficient::Get\(\)](#), [Grid\(\)](#), [SourceAndLosses::Initialize\(\)](#), [Initialize\(\)](#), [PSD::Interpolate\(\)](#), [PSD::LoadInitialValue\(\)](#), [main\(\)](#), [MakeGrid\(\)](#), and [Output\(\)](#).

8.9.4.2 int Grid::alphaSize

Definition at line 72 of file Grid.h.

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

8.9.4.3 GridElement Grid::epc

Definition at line 70 of file Grid.h.

Referenced by DiffusionCoefficient::Get(), Grid(), SourcesAndLosses::Initialize(), Initialize(), PSD::LoadInitialValue(), main(), MakeGrid(), and Output().

8.9.4.4 int Grid::epcSize

Definition at line 72 of file Grid.h.

Referenced by Initialize().

8.9.4.5 Matrix3D<double> Grid::Jacobian

Jacobian of the diffusion equation.

In 1d_universal_solver there is a parameter 'n' represented power of variable in the equation $x^n \frac{\partial}{\partial x} x^{-n} D_{xx} \frac{\partial f}{\partial x}$. More general case is using Jacobian: $Jacobian \frac{\partial}{\partial x} Jacobian^{-1} D_{xx} \frac{\partial f}{\partial x}$.

Definition at line 101 of file Grid.h.

Referenced by Grid(), main(), and MakeGrid().

8.9.4.6 GridElement Grid::L

Definition at line 70 of file Grid.h.

Referenced by DiffusionCoefficient::DiffusionCoefficient(), DiffusionCoefficient::Get(), Grid(), SourcesAndLosses::Initialize(), Initialize(), PSD::Interpolate(), PSD::LoadInitialValue(), main(), MakeGrid(), and Output().

8.9.4.7 int Grid::LSize

Definition at line 72 of file Grid.h.

Referenced by Initialize().

8.9.4.8 GridElement Grid::pc

Definition at line 70 of file Grid.h.

Referenced by DiffusionCoefficient::DiffusionCoefficient(), DiffusionCoefficient::Get(), Grid(), SourcesAndLosses::Initialize(), Initialize(), PSD::Interpolate(), PSD::LoadInitialValue(), main(), and MakeGrid().

8.9.4.9 int Grid::pcSize

Definition at line 72 of file Grid.h.

Referenced by Initialize().

8.9.4.10 string Grid::type

Definition at line 71 of file Grid.h.

Referenced by Initialize(), and PSD::Interpolate().

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

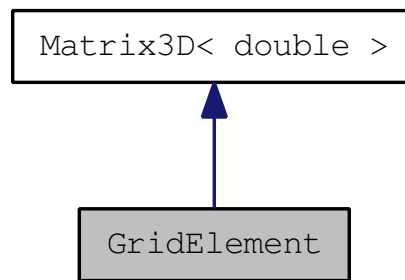
- [Grid.h](#)
- [Grid.cpp](#)

8.10 GridElement Class Reference

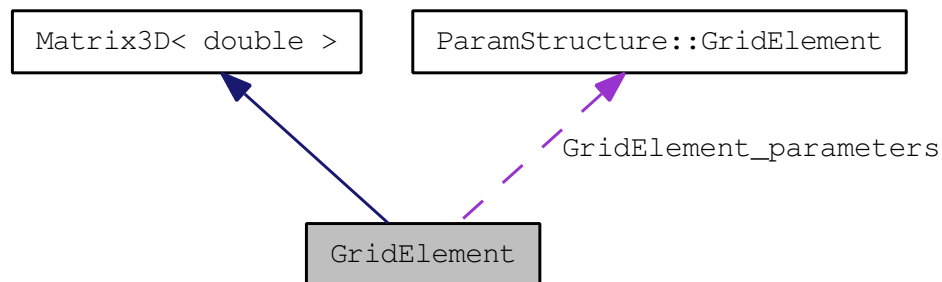
Array of values of coordinate axe.

```
#include <Grid.h>
```

Inheritance diagram for GridElement:



Collaboration diagram for GridElement:



Public Member Functions

- [GridElement](#) ()
- [GridElement](#) ([ParamStructure::GridElement](#) parameters)
- [GridElement](#) ([ParamStructure::GridElement](#) parameters, int L_size, int pc_size, int alpha_size)

Constructor.

- void [Initialize](#) ([ParamStructure::GridElement](#) parameters)
Saving parameters to the class and initializing boundary conditions.
- void [AllocateMemory](#) (int L_size, int pc_size, int alpha_size)
Allocating memory for grid element and boundaries.
- void [Initialize](#) ([ParamStructure::GridElement](#) parameters, int L_size, int pc_size, int alpha_size)
Grid element initialization.
- [GridElement](#) & [operator=](#) (const [Matrix3D< double >](#) &M)
GridElement = [Matrix3D](#).

- [GridElement](#) & [operator=](#) (double val)
GridElement = value.
- [GridElement Kfunc](#) ()
Returns epc (assumed to be applied to pc).
- void [Kfunc](#) ([GridElement](#) arg)
Convert agr to epc.
- void [SetRegularGridValue](#) (int il, int im, int ia, int gridElementDirection)
Function return the value for the specified point on regular grid.

Public Attributes

- [ParamStructure::GridElement](#) [GridElement_parameters](#)
- int [size](#)

8.10.1 Detailed Description

Array of values of coordinate axe.

Class for GridElement in parent class [Matrix3D](#) holds values of one coordinate from 3-dimentional (usually) coordinate system. Has two [BoundaryCondition](#) members: lowerBoundaryCondition and upperBoundaryCondition. Has functions to create grids of different type.

Definition at line 28 of file Grid.h.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 [GridElement::GridElement](#) () [inline]

Definition at line 39 of file Grid.h.

8.10.2.2 [GridElement::GridElement](#) ([ParamStructure::GridElement](#) *parameters*)

8.10.2.3 [GridElement::GridElement](#) ([ParamStructure::GridElement](#) *parameters*, int *L_size*, int *pc_size*, int *alpha_size*)

Constructor.

Allocate memory for grid element (allocate memory for arrays).

Parameters:

- int* L_size - L size
- int* pc_size - pc size
- int* alpha_size - alpha size Constructor.

Allocate memory for grid element (allocate memory for arrays) and run initialization.

Parameters:

ParamStructure::GridElement parameters - grid element parameters structure

int *L_size* - L size

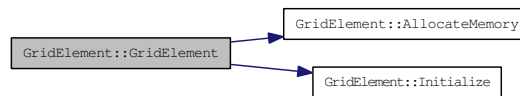
int *pc_size* - pc size

int *alpha_size* - alpha size

Definition at line 54 of file Grid.cpp.

References `AllocateMemory()`, and `Initialize()`.

Here is the call graph for this function:



8.10.3 Member Function Documentation

8.10.3.1 `void GridElement::AllocateMemory (int L_size, int pc_size, int alpha_size)`

Allocating memory for grid element and boundaries.

Parameters:

int *L_size* - L size

int *pc_size* - pc size

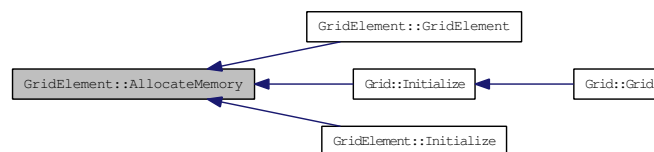
int *alpha_size* - alpha size

Reimplemented from `Matrix3D< double >`.

Definition at line 81 of file Grid.cpp.

Referenced by `GridElement()`, `Grid::Initialize()`, and `Initialize()`.

Here is the caller graph for this function:



8.10.3.2 `void GridElement::Initialize (ParamStructure::GridElement parameters, int L_size, int pc_size, int alpha_size)`

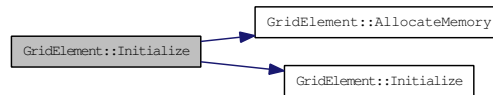
Grid element initialization.

Allocate memory for grid element (allocate memory for array) and run initialization. In case we didn't do this in constructor.

Definition at line 67 of file Grid.cpp.

References AllocateMemory(), and Initialize().

Here is the call graph for this function:



8.10.3.3 void GridElement::Initialize (ParamStructure::GridElement *parameters*)

Saving parameters to the class and initializing boundary conditions.

Parameters:

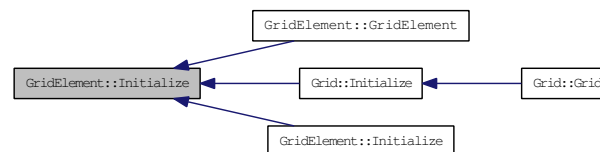
[*ParamStructure::GridElement*](#) parameters - grid element parameters structure

Definition at line 91 of file Grid.cpp.

References GridElement_parameters, ParamStructure::GridElement::name, Matrix3D< double >::name, ParamStructure::GridElement::size, and size.

Referenced by GridElement(), Grid::Initialize(), and Initialize().

Here is the caller graph for this function:



8.10.3.4 void GridElement::Kfunc (GridElement *arg*)

Convert agr to epc.

Arg should be pc.

Todo

Wired function, should be removed.

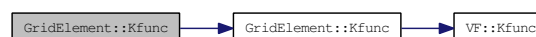
Parameters:

[*GridElement*](#) arg - should be pc.

Definition at line 139 of file Grid.cpp.

References Kfunc(), Matrix3D< double >::size_x, Matrix3D< double >::size_y, and Matrix3D< double >::size_z.

Here is the call graph for this function:



8.10.3.5 GridElement GridElement::Kfunc ()

Returns epc (assumed to be applied to pc).

Definition at line 122 of file Grid.cpp.

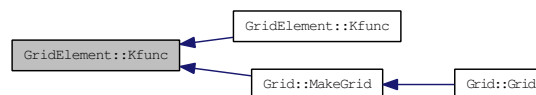
References VF::Kfunc(), Matrix3D< double >::size_x, Matrix3D< double >::size_y, and Matrix3D< double >::size_z.

Referenced by Kfunc(), and Grid::MakeGrid().

Here is the call graph for this function:



Here is the caller graph for this function:



8.10.3.6 GridElement & GridElement::operator= (double val)

GridElement = value.

Parameters:

double val - value val.

Reimplemented from [Matrix3D< double >](#).

Definition at line 113 of file Grid.cpp.

References operator=().

Here is the call graph for this function:



8.10.3.7 GridElement & GridElement::operator= (const Matrix3D< double > &M)

GridElement = [Matrix3D](#).

Parameters:

const [Matrix3D<double>](#) &M - matrix M.

Reimplemented from [Matrix3D< double >](#).

Definition at line 102 of file Grid.cpp.

Referenced by operator=().

Here is the caller graph for this function:



8.10.3.8 void GridElement::SetRegularGridValue (int iteratorL, int iteratorPc, int iteratorAlpha, int gridElementDirection)

Function return the value for the specified point on regular grid.

Very usefull function, allows to avoid a lot of typos in creation of regular grids everywhere.

Just don't have to repeat regular grid creation code a lot of times. The function create logarithmic grid if useLogScale == true as: $10^{(\log_{10}(\min) + x * (\log_{10}(\max) - \log_{10}(\min)) / (\text{size} - 1))}$; if useLogScale == false, then grid is: $\min + x * (\max - \min) / (\text{size} - 1)$. If grid size == 1 the function returns max.

PS: size, min and max stored in the gridElement already. PPS: useLogScale is also there.

Parameters:

int iteratorL - index on L-grid.

int iteratorPc - index on pc-grid.

int iteratorAlpha - index on alpha-grid.

int gridElementDirection - index on the grid, which we are making.

Definition at line 166 of file Grid.cpp.

References GridElement_parameters, ParamStructure::GridElement::max, ParamStructure::GridElement::min, ParamStructure::GridElement::size, and ParamStructure::GridElement::useLogScale.

Referenced by Grid::MakeGrid().

Here is the caller graph for this function:



8.10.4 Member Data Documentation

8.10.4.1 ParamStructure::GridElement GridElement::GridElement_parameters

Definition at line 33 of file Grid.h.

Referenced by Initialize(), PSD::Load_initial_f_2d(), Grid::MakeGrid(), and SetRegularGridValue().

8.10.4.2 int GridElement::size

Definition at line 36 of file Grid.h.

Referenced by DiffusionCoefficient::Calculate(), PSD::Diffusion_alpha(), PSD::Diffusion_L(), PSD::Diffusion_pc(), PSD::Diffusion_pc_alpha(), DiffusionCoefficient::DiffusionCoefficient(),

PSD::DiffusionMixTermExplicit(), Grid::Grid(), SourcesAndLosses::Initialize(), Grid::Initialize(), Initialize(), PSD::Interpolate(), PSD::Load_initial_f(), PSD::Load_initial_f_2d(), PSD::Load_initial_f_file(), DiffusionCoefficient::LoadDiffusionCoefficientFromFileWithGrid(), DiffusionCoefficient::LoadDiffusionCoefficientFromPlaneFile(), PSD::LoadInitialValue(), main(), DiffusionCoefficient::MakeDLL100(), DiffusionCoefficient::MakeDLL_B(), DiffusionCoefficient::MakeDLL_BE(), DiffusionCoefficient::MakeDLL_FAKE(), Grid::MakeGrid(), Grid::Output(), and PSD::SourcesAndLosses().

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

- [Grid.h](#)
- [Grid.cpp](#)

8.11 ParamStructure::GridElement Struct Reference

Grid element parameters structure.

```
#include <Parameters.h>
```

Public Attributes

- string [name](#)
Grid element name. Optional.
- int [size](#)
Grid size.
- bool [useLogScale](#)
Flag, use logarithmic scale.
- double [min](#)
min value.
- double [max](#)
max value.

8.11.1 Detailed Description

Grid element parameters structure.

Definition at line 133 of file Parameters.h.

8.11.2 Member Data Documentation

8.11.2.1 double ParamStructure::GridElement::max

max value.

Definition at line 141 of file Parameters.h.

Referenced by PSD::Load_initial_f_2d(), ParamStructure::Load_parameters(), Grid::MakeGrid(), and GridElement::SetRegularGridValue().

8.11.2.2 double ParamStructure::GridElement::min

min value.

Definition at line 140 of file Parameters.h.

Referenced by PSD::Load_initial_f_2d(), ParamStructure::Load_parameters(), Grid::MakeGrid(), and GridElement::SetRegularGridValue().

8.11.2.3 `string ParamStructure::GridElement::name`

[Grid](#) element name. Optional.

Definition at line 135 of file Parameters.h.

Referenced by `GridElement::Initialize()`, and `ParamStructure::Load_parameters()`.

8.11.2.4 `int ParamStructure::GridElement::size`

[Grid](#) size.

Definition at line 137 of file Parameters.h.

Referenced by `Grid::Initialize()`, `GridElement::Initialize()`, `ParamStructure::Load_parameters()`, and `GridElement::SetRegularGridValue()`.

8.11.2.5 `bool ParamStructure::GridElement::useLogScale`

Flag, use logarithmic scale.

Definition at line 139 of file Parameters.h.

Referenced by `ParamStructure::Load_parameters()`, `Grid::MakeGrid()`, and `GridElement::SetRegularGridValue()`.

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

- [Parameters.h](#)

8.12 ParamStructure::Interpolation Struct Reference

Interpolation parameters structure

```
#include <Parameters.h>
```

Public Attributes

- string [type](#)
Interpolation type: spline, linear etc.
- string [useLog](#)
Flag, if we should interpolate logarithm of the values.
- double [linearSplineCoef](#)
Parameters for spline interpolation - defined the ratio between spline and linear.
- double [maxSecondDerivative](#)
Maximum second derivative value, to start using linearSplineCoef.

8.12.1 Detailed Description

Interpolation parameters structure

Definition at line 210 of file Parameters.h.

8.12.2 Member Data Documentation

8.12.2.1 double ParamStructure::Interpolation::linearSplineCoef

Parameters for spline interpolation - defined the ratio between spline and linear.

0 - pure spline, 1 - pure linear.

Definition at line 217 of file Parameters.h.

Referenced by PSD::Interpolate(), and ParamStructure::Load_parameters().

8.12.2.2 double ParamStructure::Interpolation::maxSecondDerivative

Maximum second derivative value, to start using linearSplineCoef.

Definition at line 220 of file Parameters.h.

Referenced by PSD::Interpolate(), and ParamStructure::Load_parameters().

8.12.2.3 string ParamStructure::Interpolation::type

[Interpolation](#) type: spline, linear etc.

Definition at line 212 of file Parameters.h.

Referenced by PSD::Interpolate(), and ParamStructure::Load_parameters().

8.12.2.4 string ParamStructure::Interpolation::useLog

Flag, if we should interpolate logarithm of the values.

Definition at line 214 of file Parameters.h.

Referenced by PSD::Interpolate(), and ParamStructure::Load_parameters().

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

- [Parameters.h](#)

8.13 ITLIN_DATA Struct Reference

```
#include <itlin.h>
```

Public Types

- enum { [GMRES](#) = 0, [GBIT](#) = 1, [PCG](#) = 2 }
- enum { [Initial](#) = 1, [Intermediate](#) = 2, [Solution](#) = 3, [Final](#) = 4 }

Public Attributes

- double * [res](#)
- double [tau](#)
- double [t](#)
- double [normdx](#)
- double [residuum](#)
- enum ITLIN_DATA:: { ... } [codeid](#)
- enum ITLIN_DATA:: { ... } [mode](#)

8.13.1 Detailed Description

Definition at line 99 of file itlin.h.

8.13.2 Member Enumeration Documentation

8.13.2.1 anonymous enum

Enumerator:

GMRES

GBIT

PCG

Definition at line 103 of file itlin.h.

8.13.2.2 anonymous enum

Enumerator:

Initial

Intermediate

Solution

Final

Definition at line 104 of file itlin.h.

8.13.3 Member Data Documentation

8.13.3.1 `enum { ... } ITLIN_DATA::codeid`

Referenced by `gmres()`, and `itlin_dataout()`.

8.13.3.2 `enum { ... } ITLIN_DATA::mode`

Referenced by `gmres()`, and `itlin_dataout()`.

8.13.3.3 `double ITLIN_DATA::normdx`

Definition at line 102 of file `itlin.h`.

Referenced by `gmres()`, and `itlin_dataout()`.

8.13.3.4 `double* ITLIN_DATA::res`

Definition at line 101 of file `itlin.h`.

Referenced by `gmres()`, and `itlin_dataout()`.

8.13.3.5 `double ITLIN_DATA::residuum`

Definition at line 102 of file `itlin.h`.

Referenced by `gmres()`, and `itlin_dataout()`.

8.13.3.6 `double ITLIN_DATA::t`

Definition at line 102 of file `itlin.h`.

Referenced by `gmres()`, `gmres_qrfact()`, `gmres_qrsolv()`, and `itlin_dataout()`.

8.13.3.7 `double ITLIN_DATA::tau`

Definition at line 102 of file `itlin.h`.

Referenced by `gmres()`, and `itlin_dataout()`.

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

- [itlin.h](#)

8.14 ITLIN_INFO Struct Reference

```
#include <itlin.h>
```

Public Attributes

- double [precision](#)
- double [normdx](#)
- double [residuuum](#)
- int [iter](#)
- int [rcode](#)
- int [subcode](#)
- int [nomatvec](#)
- int [noprecon](#)
- int [noprecl](#)
- int [noprecr](#)

8.14.1 Detailed Description

Definition at line 93 of file itlin.h.

8.14.2 Member Data Documentation

8.14.2.1 int ITLIN_INFO::iter

Definition at line 96 of file itlin.h.

Referenced by [gmres\(\)](#), and [gmres_wrapout\(\)](#).

8.14.2.2 int ITLIN_INFO::nomatvec

Definition at line 96 of file itlin.h.

Referenced by [gmres\(\)](#), and [gmres_wrapout\(\)](#).

8.14.2.3 int ITLIN_INFO::noprecl

Definition at line 96 of file itlin.h.

Referenced by [gmres\(\)](#), and [gmres_wrapout\(\)](#).

8.14.2.4 int ITLIN_INFO::noprecon

Definition at line 96 of file itlin.h.

8.14.2.5 int ITLIN_INFO::noprecr

Definition at line 96 of file itlin.h.

Referenced by [gmres\(\)](#), and [gmres_wrapout\(\)](#).

8.14.2.6 double ITLIN_INFO::normdx

Definition at line 95 of file itlin.h.

8.14.2.7 double ITLIN_INFO::precision

Definition at line 95 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.14.2.8 int ITLIN_INFO::rcode

Definition at line 96 of file itlin.h.

Referenced by gmres_wrapout().

8.14.2.9 double ITLIN_INFO::residuum

Definition at line 95 of file itlin.h.

8.14.2.10 int ITLIN_INFO::subcode

Definition at line 96 of file itlin.h.

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

- [itlin.h](#)

8.15 ITLIN_IO Struct Reference

```
#include <itlin.h>
```

Public Attributes

- FILE * [errfile](#)
- FILE * [monfile](#)
- FILE * [datfile](#)
- FILE * [iterfile](#)
- FILE * [resfile](#)
- FILE * [miscfile](#)
- [PRINT_LEVEL](#) [errlevel](#)
- [PRINT_LEVEL](#) [monlevel](#)
- [PRINT_LEVEL](#) [datlevel](#)

8.15.1 Detailed Description

Definition at line 107 of file itlin.h.

8.15.2 Member Data Documentation

8.15.2.1 FILE * ITLIN_IO::datfile

Definition at line 109 of file itlin.h.

8.15.2.2 PRINT_LEVEL ITLIN_IO::datlevel

Definition at line 111 of file itlin.h.

8.15.2.3 FILE* ITLIN_IO::errfile

Definition at line 109 of file itlin.h.

8.15.2.4 PRINT_LEVEL ITLIN_IO::errlevel

Definition at line 111 of file itlin.h.

8.15.2.5 FILE * ITLIN_IO::iterfile

Definition at line 109 of file itlin.h.

8.15.2.6 FILE * ITLIN_IO::miscfile

Definition at line 109 of file itlin.h.

8.15.2.7 FILE * ITLIN_IO::monfile

Definition at line 109 of file itlin.h.

8.15.2.8 PRINT_LEVEL ITLIN_IO::monlevel

Definition at line 111 of file itlin.h.

8.15.2.9 FILE * ITLIN_IO::resfile

Definition at line 109 of file itlin.h.

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

- [itlin.h](#)

8.16 ITLIN_OPT Struct Reference

```
#include <itlin.h>
```

Public Attributes

- double [tol](#)
- double [rho](#)
- int [i_max](#)
- int [maxiter](#)
- [TERM_CHECK](#) [termcheck](#)
- [CONV_CHECK](#) [convcheck](#)
- [LOGICAL](#) [rescale](#)
- [PRINT_LEVEL](#) [errorlevel](#)
- [PRINT_LEVEL](#) [monitorlevel](#)
- [PRINT_LEVEL](#) [datalevel](#)
- FILE * [errorfile](#)
- FILE * [monitorfile](#)
- FILE * [datafile](#)
- FILE * [iterfile](#)
- FILE * [resfile](#)
- FILE * [miscfile](#)
- double * [scale](#)

8.16.1 Detailed Description

Definition at line 80 of file `itlin.h`.

8.16.2 Member Data Documentation

8.16.2.1 [CONV_CHECK](#) `ITLIN_OPT::convcheck`

Definition at line 85 of file `itlin.h`.

Referenced by `itlin_parcheck_and_print()`.

8.16.2.2 [FILE *](#) `ITLIN_OPT::datafile`

Definition at line 88 of file `itlin.h`.

Referenced by `gmres()`, and `gmres_wrapout()`.

8.16.2.3 [PRINT_LEVEL](#) `ITLIN_OPT::datalevel`

Definition at line 87 of file `itlin.h`.

Referenced by `gmres()`, and `gmres_wrapout()`.

8.16.2.4 FILE* ITLIN_OPT::errorfile

Definition at line 88 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.5 PRINT_LEVEL ITLIN_OPT::errorlevel

Definition at line 87 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.6 int ITLIN_OPT::i_max

Definition at line 83 of file itlin.h.

Referenced by gmres(), gmres_wrapout(), and itlin_parcheck_and_print().

8.16.2.7 FILE * ITLIN_OPT::iterfile

Definition at line 88 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.8 int ITLIN_OPT::maxiter

Definition at line 83 of file itlin.h.

Referenced by gmres(), gmres_wrapout(), and itlin_parcheck_and_print().

8.16.2.9 FILE * ITLIN_OPT::miscfile

Definition at line 88 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.10 FILE * ITLIN_OPT::monitorfile

Definition at line 88 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.11 PRINT_LEVEL ITLIN_OPT::monitorlevel

Definition at line 87 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.12 LOGICAL ITLIN_OPT::rescale

Definition at line 86 of file itlin.h.

8.16.2.13 FILE * ITLIN_OPT::resfile

Definition at line 88 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.14 double ITLIN_OPT::rho

Definition at line 82 of file itlin.h.

Referenced by itlin_parcheck_and_print().

8.16.2.15 double* ITLIN_OPT::scale

Definition at line 90 of file itlin.h.

Referenced by itlin_parcheck_and_print().

8.16.2.16 TERM_CHECK ITLIN_OPT::termcheck

Definition at line 84 of file itlin.h.

Referenced by gmres(), and gmres_wrapout().

8.16.2.17 double ITLIN_OPT::tol

Definition at line 82 of file itlin.h.

Referenced by gmres(), gmres_wrapout(), and itlin_parcheck_and_print().

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

- [itlin.h](#)

8.17 Maths::Interpolation::Linear Class Reference

[Linear](#) interpolation.

```
#include <linear.h>
```

Public Member Functions

- [Linear](#) (int *n*, double **x*, double **y*)
- [~Linear](#) ()
- double [getValue](#) (double *x*)

Public Attributes

- int [size](#)

8.17.1 Detailed Description

[Linear](#) interpolation.

Definition at line 15 of file linear.h.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 Maths::Interpolation::Linear::Linear (int *n*, double **x*, double **y*) [inline]

Definition at line 19 of file linear.h.

References [i](#), and [size](#).

8.17.2.2 Maths::Interpolation::Linear::~~Linear () [inline]

Definition at line 31 of file linear.h.

8.17.3 Member Function Documentation

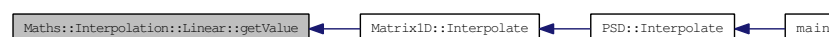
8.17.3.1 double Maths::Interpolation::Linear::getValue (double *x*) [inline]

Definition at line 38 of file linear.h.

References [i](#), and [size](#).

Referenced by [Matrix1D< T >::Interpolate\(\)](#).

Here is the caller graph for this function:



8.17.4 Member Data Documentation

8.17.4.1 int Maths::Interpolation::Linear::size

Definition at line 17 of file linear.h.

Referenced by `getValue()`, and `Linear()`.

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

- [linear.h](#)

8.18 Matrix1D< T > Class Template Reference

Matrix 1D class.

```
#include <Matrix.h>
```

Public Member Functions

- [Matrix1D](#) ()
Default constructor - do nothing.
- [Matrix1D](#) (int [size_x](#))
Constructor.
- [Matrix1D](#) (int [x_size](#), string [name](#))
Constructor.
- [Matrix1D](#) (const [Matrix1D](#)< T > &M)
Constructor.
- [~Matrix1D](#) ()
Destructor.
- void [AllocateMemory](#) (int [size_x](#))
Allocating memory.
- T [max](#) ()
- T [maxabs](#) ()
- T & [operator\[\]](#) (int i)
Return the i-th value of matrix.
- T & [operator\(\)](#) (int x)
Return the x-th value of matrix.
- [Matrix1D](#) & [operator=](#) (const [Matrix1D](#)< T > &M)
Make matrix equal to matrix M Return the same instance of class Matrix.
- [Matrix1D](#) & [operator=](#) (T Val)
Make matrix equal to value Val.
- [Matrix1D](#) [operator/](#) (T Val)
Divide a matrix to a value Val.
- [Matrix1D](#) [operator*](#) (T Val)
Multiply a matrix to a value Val.
- [Matrix1D](#) [operator/](#) ([Matrix1D](#)< T > &M)
Divide all values of matrix to values of matrix M.
- [Matrix1D](#) [operator*](#) ([Matrix1D](#)< T > &M)

Multiply all values of matrix to values of matrix M.

- void [Interpolate](#) ([Matrix1D](#)< T > &old_function, [Matrix1D](#)< T > &old_grid, [Matrix1D](#)< T > &new_grid)

Linear interpolation.

- void [Spline](#) ([Matrix1D](#)< T > &old_function, [Matrix1D](#)< T > &old_grid, [Matrix1D](#)< T > &new_grid, double lb, double ub, double lin_spline_coef=0, double max_second_der=0)

Spline interpolation.

- void [Polint](#) ([Matrix1D](#)< T > &old_function, [Matrix1D](#)< T > &old_grid, [Matrix1D](#)< T > &new_grid)

polynomial Some other wired interpolation attempts.

- void [Ratint](#) ([Matrix1D](#)< T > &old_function, [Matrix1D](#)< T > &old_grid, [Matrix1D](#)< T > &new_grid)

- void [Polilinear](#) ([Matrix1D](#)< T > &old_function, [Matrix1D](#)< T > &old_grid, [Matrix1D](#)< T > &new_grid, double lb, double ub)

polynomial + lin Some other wired interpolation attempts.

- void [writeToFile](#) (string filename)

Write matrix data to file.

- void [writeToFile](#) (string filename, [Matrix1D](#)< T > &grid_x)

Write matrix data to file with grid.

- void [readFromFile](#) (string filename)

Read matrix data from file.

- void [readFromFile](#) (string filename, [Matrix1D](#)< T > &grid_x)

Read matrix data from file and check 'grid'.

Public Attributes

- bool [initialized](#)

Flag, equal true if initialized.

- int [size_x](#)

size x

- string [name](#)

name of the Matrix

8.18.1 Detailed Description

template<typename T> class Matrix1D< T >

Matrix 1D class.

Matrixes and operations.

Definition at line 38 of file Matrix.h.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 `template<typename T> Matrix1D< T >::Matrix1D ()` [inline]

Default constructor - do nothing.

Definition at line 49 of file Matrix.h.

8.18.2.2 `template<class T > Matrix1D< T >::Matrix1D (int x_size)` [inline]

Constructor.

Runs allocating memory function.

Parameters:

int *x_size* - size of the matrix

Definition at line 142 of file Matrix.cpp.

References `Matrix1D< T >::AllocateMemory()`, and `Matrix1D< T >::initialized`.

Here is the call graph for this function:



8.18.2.3 `template<class T > Matrix1D< T >::Matrix1D (int x_size, string name)` [inline]

Constructor.

Runs allocating memory function and store matrix name.

Parameters:

int *x_size* - size of the matrix

string *name* - name of the matrix

Definition at line 126 of file Matrix.cpp.

References `Matrix1D< T >::AllocateMemory()`, and `Matrix1D< T >::initialized`.

Here is the call graph for this function:



8.18.2.4 `template<class T> Matrix1D< T >::Matrix1D (const Matrix1D< T > & M)` [inline]

Constructor.

Make new matrix equal to Matrix M.

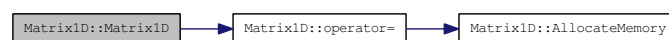
Parameters:

const Matrix1D<T> &M - matrix M

Definition at line 157 of file Matrix.cpp.

References Matrix1D< T >::initialized, and Matrix1D< T >::operator=().

Here is the call graph for this function:



8.18.2.5 `template<class T> Matrix1D< T >::~~Matrix1D ()` [inline]

Destructor.

Destruct the class.

Definition at line 166 of file Matrix.cpp.

References Matrix1D< T >::initialized.

8.18.3 Member Function Documentation

8.18.3.1 `template<class T> void Matrix1D< T >::AllocateMemory (int x_size)` [inline]

Allocating memory.

Parameters:

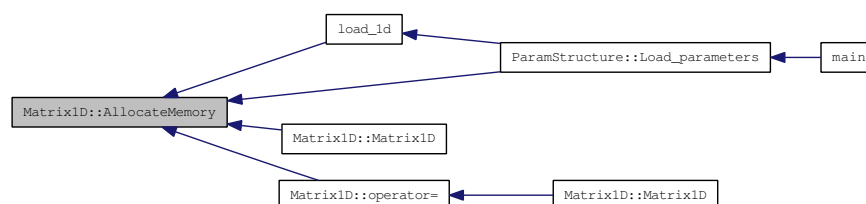
int x_size - size x

Definition at line 176 of file Matrix.cpp.

References Matrix1D< T >::initialized, and Matrix1D< T >::size_x.

Referenced by load_ld(), ParamStructure::Load_parameters(), Matrix1D< T >::Matrix1D(), and Matrix1D< T >::operator=().

Here is the caller graph for this function:



8.18.3.2 `template<class T> void Matrix1D< T >::Interpolate (Matrix1D< T > &old_function, Matrix1D< T > &old_grid, Matrix1D< T > &new_grid) [inline]`

Linear interpolation.

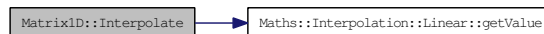
Runs interpolation function from Numerical Recepties.

Definition at line 1259 of file Matrix.cpp.

References Maths::Interpolation::Linear::getValue(), i, and Matrix1D< T >::size_x.

Referenced by PSD::Interpolate().

Here is the call graph for this function:



Here is the caller graph for this function:



8.18.3.3 `template<typename T> T Matrix1D< T >::max ()`

8.18.3.4 `template<typename T> T Matrix1D< T >::maxabs ()`

8.18.3.5 `template<class T> T & Matrix1D< T >::operator() (int x) [inline]`

Return the x-th value of matrix.

Operator (x), returns value of element x.

If DEBUG_MODE defined, check if matrix has been initialized. No difference between [] and () operators for 1d-matrix class.

Parameters:

x - number of element to return

Definition at line 220 of file Matrix.cpp.

References Matrix1D< T >::initialized.

8.18.3.6 `template<class T> Matrix1D< T > Matrix1D< T >::operator* (Matrix1D< T > &M) [inline]`

Multiply all values of matrix to values of matrix M.

Parameters:

Matrix1D<T> &M - matrix M.

Definition at line 321 of file Matrix.cpp.

References Matrix1D< T >::size_x.

8.18.3.7 `template<class T> Matrix1D< T > Matrix1D< T >::operator* (T Val)` [inline]

Multiply a matrix to a value Val.

Return new instance of class Matrix.

Parameters:

T Val - value Val

Definition at line 293 of file Matrix.cpp.

References Matrix1D< T >::size_x.

8.18.3.8 `template<class T> Matrix1D< T > Matrix1D< T >::operator/ (Matrix1D< T > &M)` [inline]

Divide all values of matrix to values of matrix M.

Parameters:

Matrix1D<T> &M - matrix M.

Definition at line 307 of file Matrix.cpp.

References Matrix1D< T >::size_x.

8.18.3.9 `template<class T> Matrix1D< T > Matrix1D< T >::operator/ (T Val)` [inline]

Divide a matrix to a value Val.

Return new instance of class Matrix.

Parameters:

T Val - value Val

Definition at line 278 of file Matrix.cpp.

References Matrix1D< T >::size_x.

8.18.3.10 `template<class T> Matrix1D< T > & Matrix1D< T >::operator= (T Val)` [inline]

Make matrix equal to value Val.

Return the same instance of class Matrix.

Parameters:

T Val - value val

Definition at line 265 of file Matrix.cpp.

References Matrix1D< T >::size_x.

8.18.3.11 `template<class T> Matrix1D< T > & Matrix1D< T >::operator= (const Matrix1D< T > & M) [inline]`

Make matrix equal to matrix M Return the same instance of class Matrix.

Parameters:

const Matrix1D<T> &M - matrix M

Definition at line 237 of file Matrix.cpp.

References Matrix1D< T >::AllocateMemory(), Matrix1D< T >::initialized, Matrix1D< T >::name, and Matrix1D< T >::size_x.

Referenced by Matrix1D< T >::Matrix1D().

Here is the call graph for this function:



Here is the caller graph for this function:



8.18.3.12 `template<class T> T & Matrix1D< T >::operator[] (int i) [inline]`

Return the i-th value of matrix.

Operator [i], returns value of element i.

If DEBUG_MODE defined, check if matrix has been initialized.

Parameters:

i - number of element to return

Definition at line 192 of file Matrix.cpp.

References Matrix1D< T >::initialized.

8.18.3.13 `template<class T> void Matrix1D< T >::Polilinear (Matrix1D< T > & old_function, Matrix1D< T > & old_grid, Matrix1D< T > & new_grid, double lb, double ub) [inline]`

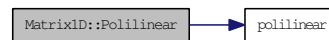
polynomial + lin Some other wired interpolation attempts.

Definition at line 1451 of file Matrix.cpp.

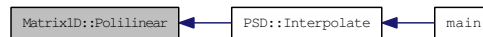
References i, polilinear(), and Matrix1D< T >::size_x.

Referenced by PSD::Interpolate().

Here is the call graph for this function:



Here is the caller graph for this function:



8.18.3.14 `template<class T> void Matrix1D< T >::Polint (Matrix1D< T > & old_function, Matrix1D< T > & old_grid, Matrix1D< T > & new_grid) [inline]`

polynomial Some other wired interpolation attempts.

Definition at line 1469 of file Matrix.cpp.

References `err`, `i`, `polint()`, and `Matrix1D< T >::size_x`.

Here is the call graph for this function:

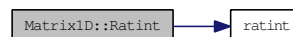


8.18.3.15 `template<class T> void Matrix1D< T >::Ratint (Matrix1D< T > & old_function, Matrix1D< T > & old_grid, Matrix1D< T > & new_grid) [inline]`

Definition at line 1485 of file Matrix.cpp.

References `err`, `i`, `ratint()`, and `Matrix1D< T >::size_x`.

Here is the call graph for this function:



8.18.3.16 `template<class T> void Matrix1D< T >::readFromFile (string filename, Matrix1D< T > & grid_x) [inline]`

Read matrix data from file and check 'grid'.

Todo

All matrix readings change to [readFromFile\(\)](#) function

Definition at line 405 of file Matrix.cpp.

References `err`, `Matrix1D< T >::initialized`, and `Matrix1D< T >::size_x`.

8.18.3.17 `template<class T> void Matrix1D< T >::readFromFile (string filename)` [inline]

Read matrix data from file.

Definition at line 372 of file Matrix.cpp.

References `Matrix1D< T >::initialized`, and `Matrix1D< T >::size_x`.

8.18.3.18 `template<class T> void Matrix1D< T >::Spline (Matrix1D< T > &old_function, Matrix1D< T > &old_grid, Matrix1D< T > &new_grid, double lb, double ub, double lin_spline_coef = 0, double max_second_der = 0)` [inline]

Spline interpolation.

Runs interpolation functions from Numerical Recipes. Treats boundary conditions separate, i.e. use boundary values on boundary conditions and extrapolation, use linear interpolation for next-to-the-boundary points. Has additional mechanism of smoothing the interpolation - makes it more linear according the parameters `lin_spline_coef` and `max_second_der`. Spline interpolation is linear + additional terms from second derivatives (look Numerical Recipes). If second derivative > `max_second_der` the second derivatives are multiplied by `lin_spline_coef`.

Parameters:

Matrix1D<T> &old_function - old function

Matrix1D<T> &old_grid - old grid

Matrix1D<T> &new_grid - new grid

double lb - low boundary value

double ub - upper boundary value

double lin_spline_coef - coefficient to multiply the second derivatives terms (makes interpolation more smooth)

double max_second_der - maximum second derivatives, all greater derivatives gonna be multiplied by `lin_spline_coef`

Todo

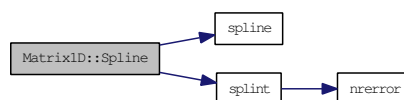
Check the index of the last argument of the spline interpolation.

Definition at line 1300 of file Matrix.cpp.

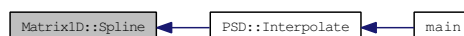
References `i`, `Matrix1D< T >::size_x`, `spline()`, and `splint()`.

Referenced by `PSD::Interpolate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.18.3.19 `template<class T> void Matrix1D< T >::writeToFile (string filename, Matrix1D< T > & grid_x) [inline]`

Write matrix data to file with grid.

Definition at line 352 of file Matrix.cpp.

References Matrix1D< T >::name, and Matrix1D< T >::size_x.

8.18.3.20 `template<class T> void Matrix1D< T >::writeToFile (string filename) [inline]`

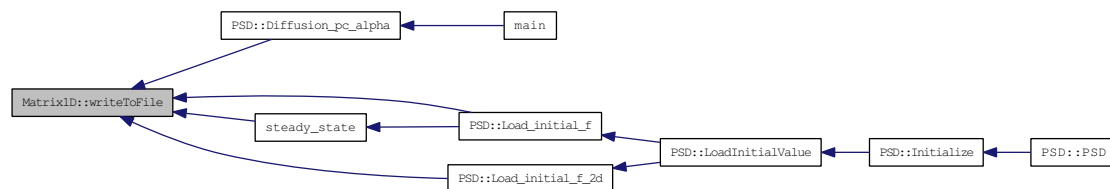
Write matrix data to file.

Definition at line 333 of file Matrix.cpp.

References Matrix1D< T >::name, and Matrix1D< T >::size_x.

Referenced by PSD::Diffusion_pc_alpha(), PSD::Load_initial_f(), PSD::Load_initial_f_2d(), and steady_state().

Here is the caller graph for this function:



8.18.4 Member Data Documentation

8.18.4.1 `template<typename T> bool Matrix1D< T >::initialized`

Flag, equal true if initialized.

Definition at line 43 of file Matrix.h.

Referenced by Matrix1D< T >::AllocateMemory(), Matrix1D< double >::Matrix1D(), Matrix1D< T >::Matrix1D(), Matrix1D< T >::operator(), Matrix1D< T >::operator=(), Matrix1D< T >::operator[](), Matrix1D< T >::readFromFile(), and Matrix1D< T >::~~Matrix1D().

8.18.4.2 `template<typename T> string Matrix1D< T >::name`

name of the Matrix

Definition at line 45 of file Matrix.h.

Referenced by Matrix1D< T >::operator=(), and Matrix1D< T >::writeToFile().

8.18.4.3 `template<typename T> int Matrix1D< T >::size_x`

size x

Definition at line 44 of file Matrix.h.

Referenced by `Matrix1D< T >::AllocateMemory()`, `Matrix1D< T >::Interpolate()`, `load_1d()`, `PSD::Load_initial_f()`, `Matrix1D< T >::operator*`, `Matrix1D< T >::operator/()`, `Matrix1D< T >::operator=()`, `Matrix1D< T >::Polilinear()`, `Matrix1D< T >::Polint()`, `Matrix1D< T >::Ratint()`, `Matrix1D< T >::readFromFile()`, `Matrix1D< T >::Spline()`, and `Matrix1D< T >::writeToFile()`.

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

- [Matrix.h](#)
- [Matrix.cpp](#)

8.19 Matrix2D< T > Class Template Reference

Matrix 2D class.

```
#include <Matrix.h>
```

Public Member Functions

- `int index1d (int x, int y)`
Returns corresponding index of 1d array.
- `Matrix2D ()`
- `Matrix2D (const Matrix2D< T > &M)`
Constructor.
- `Matrix2D (int size_x, int size_y)`
Constructor.
- `~Matrix2D ()`
Destructor.
- `void AllocateMemory (int size_x, int size_y)`
Allocate memory.
- `T * operator[] (int i)`
Return the i-th pointer to 1d-array. Next [j] can be applied, so we have regular [i][j].
- `Matrix2D & operator= (const Matrix2D< T > &M)`
*Return the (x,y)-th value of matrix !! T operator()(int x, int y) { return matrix_array[x*size_y + y]; }.*
- `Matrix2D & operator= (T val)`
Make matrix equal to value Val.
- `Matrix2D operator/ (T Val)`
Divide matrix to Val.
- `Matrix2D operator* (T Val)`
Multiply matrix to Val.
- `Matrix2D operator/ (Matrix2D< T > &M)`
Divide all values of matrix to values of matrix M.
- `Matrix2D operator* (Matrix2D< T > &M)`
Multiply all values of matrix to values of matrix M.
- `Matrix2D max (T val)`
Divide all values of matrix to value Val.
- `void Interpolate (Matrix2D< T > &old_function, Matrix2D< T > &old_grid_x, Matrix2D< T > &old_grid_y, Matrix2D< T > &new_grid_x, Matrix2D< T > &new_grid_y)`

Linear 2D interpolation.

- void `writeToFile` (string filename)
Writes the matrix to a file.
- void `writeToFile` (string filename, `Matrix2D< T > &grid_x`, `Matrix2D< T > &grid_y`)
Write the matrix to a file using other two matrixes as a grid.
- void `readFromFile` (string filename)
Read matrix data from file.
- void `readFromFile` (string filename, `Matrix2D< T > &grid_x`, `Matrix2D< T > &grid_y`)
Read matrix data from file and check grid.

Public Attributes

- bool `initialized`
Flag, equal true if initialized.
- int `size_x`
- int `size_y`
size x, size_y
- string `name`
name of the Matrix

8.19.1 Detailed Description

`template<typename T> class Matrix2D< T >`

Matrix 2D class.

Matrixes and operations.

Definition at line 93 of file Matrix.h.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 `template<typename T> Matrix2D< T >::Matrix2D ()` `[inline]`

Definition at line 108 of file Matrix.h.

References `Matrix2D< T >::initialized`.

8.19.2.2 `template<class T > Matrix2D< T >::Matrix2D (const Matrix2D< T > & M)`
`[inline]`

Constructor.

Create new matrix from the Matrix M.

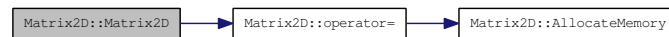
Parameters:

const Matrix2D<T> &M - Matrix M

Definition at line 471 of file Matrix.cpp.

References Matrix2D< T >::initialized, and Matrix2D< T >::operator=().

Here is the call graph for this function:



8.19.2.3 `template<class T> Matrix2D< T >::Matrix2D (int x_size, int y_size) [inline]`

Constructor.

Allocate memory.

Parameters:

int x_size - x size

int y_size - y size

Definition at line 458 of file Matrix.cpp.

References Matrix2D< T >::AllocateMemory(), and Matrix2D< T >::initialized.

Here is the call graph for this function:



8.19.2.4 `template<class T> Matrix2D< T >::~~Matrix2D () [inline]`

Destructor.

Definition at line 480 of file Matrix.cpp.

References Matrix2D< T >::initialized.

8.19.3 Member Function Documentation

8.19.3.1 `template<class T> void Matrix2D< T >::AllocateMemory (int x_size, int y_size) [inline]`

Allocate memory.

Parameters:

int x_size - x size

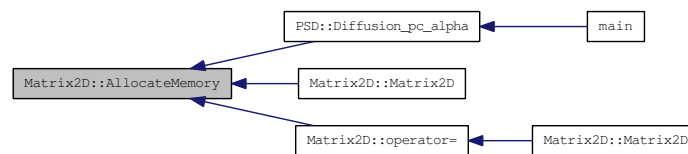
int y_size - y size

Definition at line 491 of file Matrix.cpp.

References Matrix2D< T >::initialized, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

Referenced by PSD::Diffusion_pc_alpha(), Matrix2D< T >::Matrix2D(), and Matrix2D< T >::operator=().

Here is the caller graph for this function:



8.19.3.2 `template<class T> int Matrix2D< T >::index1d(int x, int y) [inline]`

Returns corresponding index of 1d array.

Definition at line 506 of file Matrix.cpp.

References Matrix2D< T >::size_y.

8.19.3.3 `template<class T> void Matrix2D< T >::Interpolate(Matrix2D< T > &old_function, Matrix2D< T > &old_grid_x, Matrix2D< T > &old_grid_y, Matrix2D< T > &new_grid_x, Matrix2D< T > &new_grid_y) [inline]`

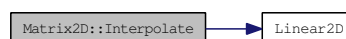
Linear 2D interpolation.

Not sure if it is working.

Definition at line 1548 of file Matrix.cpp.

References i, Linear2D(), Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

Here is the call graph for this function:



8.19.3.4 `template<class T> Matrix2D< T > Matrix2D< T >::max(T val) [inline]`

Divide all values of matrix to value Val.

Parameters:

Matrix2D<T> &M - matrix M.

Definition at line 625 of file Matrix.cpp.

References Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.5 template<class T > Matrix2D< T > Matrix2D< T >::operator* (Matrix2D< T > &M)
[inline]

Multiply all values of matrix to values of matrix M.

Parameters:

Matrix2D<T> &M - matrix M.

Definition at line 610 of file Matrix.cpp.

References Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.6 template<class T > Matrix2D< T > Matrix2D< T >::operator* (T Val) [inline]

Multiply matrix to Val.

Parameters:

T val - value Val.

Definition at line 580 of file Matrix.cpp.

References Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.7 template<class T > Matrix2D< T > Matrix2D< T >::operator/ (Matrix2D< T > &M)
[inline]

Divide all values of matrix to values of matrix M.

Parameters:

Matrix2D<T> &M - matrix M.

Definition at line 595 of file Matrix.cpp.

References Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.8 template<class T > Matrix2D< T > Matrix2D< T >::operator/ (T Val) [inline]

Divide matrix to Val.

Parameters:

T val - value Val.

Definition at line 565 of file Matrix.cpp.

References Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.9 template<class T > Matrix2D< T > & Matrix2D< T >::operator= (T val) [inline]

Make matrix equal to value Val.

Parameters:

T val - value Val.

Definition at line 546 of file Matrix.cpp.

References Matrix2D< T >::initialized, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.10 `template<class T> Matrix2D< T> & Matrix2D< T>::operator= (const Matrix2D< T> & M) [inline]`

Return the (x,y)-th value of matrix !! T operator()(int x, int y) { return matrix_array[x*size_y + y]; }.

Make matrix equal to Matrix M.

Parameters:

const Matrix2D<T> &M - Matrix M.

Definition at line 517 of file Matrix.cpp.

References Matrix2D< T >::AllocateMemory(), Matrix2D< T >::initialized, Matrix2D< T >::name, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

Referenced by Matrix2D< T >::Matrix2D().

Here is the call graph for this function:



Here is the caller graph for this function:



8.19.3.11 `template<typename T> T* Matrix2D< T>::operator[] (int i) [inline]`

Return the i-th pointer to 1d-array. Next [j] can be applied, so we have regular [i][j].

Definition at line 117 of file Matrix.h.

8.19.3.12 `template<class T> void Matrix2D< T>::readFromFile (string filename, Matrix2D< T> & grid_x, Matrix2D< T> & grid_y) [inline]`

Read matrix data from file and check grid.

Definition at line 718 of file Matrix.cpp.

References err, Matrix2D< T >::initialized, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.13 `template<class T> void Matrix2D< T>::readFromFile (string filename) [inline]`

Read matrix data from file.

Definition at line 683 of file Matrix.cpp.

References Matrix2D< T >::initialized, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.14 `template<class T> void Matrix2D< T >::writeToFile (string filename, Matrix2D< T > & grid_x, Matrix2D< T > & grid_y)` [inline]

Write the matrix to a file using other two matrixes as a grid.

Simply that means - write all three matrixes to a file.

Definition at line 665 of file Matrix.cpp.

References Matrix2D< T >::name, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.3.15 `template<class T> void Matrix2D< T >::writeToFile (string filename)` [inline]

Writes the matrix to a file.

File has two header lines.

Todo

Change the name 'writeToFile' in Matrix classes to 'write' or something.

Parameters:

filename - file name

Definition at line 643 of file Matrix.cpp.

References Matrix2D< T >::name, Matrix2D< T >::size_x, and Matrix2D< T >::size_y.

8.19.4 Member Data Documentation

8.19.4.1 `template<typename T> bool Matrix2D< T >::initialized`

Flag, equal true if initialized.

Definition at line 100 of file Matrix.h.

Referenced by Matrix2D< T >::AllocateMemory(), PSD::Diffusion_pc_alpha(), PSD::LoadInitialValue(), Matrix2D< T >::Matrix2D(), Matrix3D< T >::operator=(), Matrix2D< T >::operator=(), Matrix2D< T >::readFromFile(), and Matrix2D< T >::~Matrix2D().

8.19.4.2 `template<typename T> string Matrix2D< T >::name`

name of the Matrix

Definition at line 102 of file Matrix.h.

Referenced by Matrix2D< T >::operator=(), BoundaryCondition::Update(), Matrix2D< T >::writeToFile(), Matrix3D< T >::xSlice(), Matrix3D< T >::ySlice(), and Matrix3D< T >::zSlice().

8.19.4.3 `template<typename T> int Matrix2D< T >::size_x`

Definition at line 101 of file Matrix.h.

Referenced by `Matrix2D< T >::AllocateMemory()`, `Matrix2D< T >::Interpolate()`, `Matrix2D< T >::max()`, `Matrix2D< T >::operator*()`, `Matrix2D< T >::operator/()`, `Matrix3D< T >::operator=()`, `Matrix2D< T >::operator=()`, `Matrix2D< T >::readFromFile()`, and `Matrix2D< T >::writeToFile()`.

8.19.4.4 `template<typename T> int Matrix2D< T >::size_y`

size x, size_y

Definition at line 101 of file Matrix.h.

Referenced by `Matrix2D< T >::AllocateMemory()`, `Matrix2D< T >::index1d()`, `Matrix2D< T >::Interpolate()`, `Matrix2D< T >::max()`, `Matrix2D< T >::operator*()`, `Matrix2D< T >::operator/()`, `Matrix3D< T >::operator=()`, `Matrix2D< T >::operator=()`, `Matrix2D< T >::readFromFile()`, and `Matrix2D< T >::writeToFile()`.

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

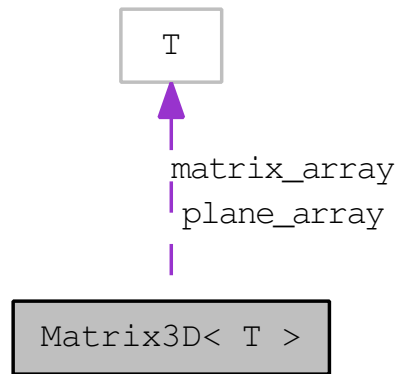
- [Matrix.h](#)
- [Matrix.cpp](#)

8.20 Matrix3D< T > Class Template Reference

Matrix 3D class.

```
#include <Matrix.h>
```

Collaboration diagram for Matrix3D< T >:



Public Member Functions

- `int index1d (int x, int y, int z)`
Returns corresponding index of 1d array.
- `Matrix3D ()`
Default constructor. Do nothing.
- `Matrix3D (const Matrix3D< T > &M)`
Constructor.
- `Matrix3D (int size_x, int size_y, int size_z)`
Constructor.
- `~Matrix3D ()`
Destructor.
- `void AllocateMemory (int size_x, int size_y, int size_z)`
Alocating memory and filling it with zero-valiues.
- `T max ()`
Return maximum value of the matrix.
- `T maxabs ()`
Return absolute maximum value of the matrix.
- `Matrix3D< T > abs ()`
Return absolute value of the matrix.

- [Matrix2D< T > xSlice](#) (int p_x)
Make x-slice of 3d matrix - 2d matrix.
- [Matrix2D< T > ySlice](#) (int p_y)
Make y-slice of 3d matrix - 2d matrix.
- [Matrix2D< T > zSlice](#) (int p_z)
Make z-slice of 3d matrix - 2d matrix.
- [T ** operator\[\]](#) (int i)
Return the i-th pointer to 2d-array. Next [j][k] can be applied, so we have regular [i][j][k].
- [T & operator\(\)](#) (int x, int y, int z)
Return the (x,y,z) value of matrix.
- [T & Value](#) (int x, int y, int z)
- [Matrix3D & operator=](#) (const [Matrix3D< T > &M](#))
Makes matrix equal to Matrix M.
- [Matrix3D & operator=](#) ([Matrix2D< T > &M](#))
Makes 3D matrix from 2D matrix.
- [Matrix3D & operator=](#) (T Val)
Makes Matrix equal to value Val.
- [Matrix3D operator/](#) (T Val)
Divide each element of the matrix to Val.
- [Matrix3D operator*](#) (T Val)
Multiply eqach element of the matrix to Val.
- [Matrix3D operator/](#) ([Matrix3D< T > &M](#))
Divide each element of the matrix to corresponds element of matrix M.
- [Matrix3D operator*](#) ([Matrix3D< T > &M](#))
Multiply each element of the matrix to corresponds element of matrix M.
- [Matrix3D operator+](#) ([Matrix3D< T > &M](#))
Add each element of the matrix to corresponds element of matrix M.
- [Matrix3D operator-](#) ([Matrix3D< T > &M](#))
Substract each element of the matrix to corresponds element of matrix M.
- void [writeToFile](#) (string filename)
Write matrix to file.
- void [writeToFile](#) (string filename, [Matrix3D< T > &grid_x](#), [Matrix3D< T > &grid_y](#), [Matrix3D< T > &grid_z](#))
Write matrix to file, using 3 other matrixes as a grid (simply - write all 4 matrixes to the file).

- void [readFromFile](#) (string filename)
Read matrix data from file.
- void [readFromFile](#) (string filename, [Matrix3D< T >](#) &grid_x, [Matrix3D< T >](#) &grid_y, [Matrix3D< T >](#) &grid_z)
Read matrix data from file with grid.

Public Attributes

- bool [initialized](#)
Flag, equal true if initialized.
- string [change_ind](#)
Variables useful for changes tracking.
- int [size_x](#)
- int [size_y](#)
- int [size_z](#)
size x, size y, size z
- string [name](#)
name of the Matrix

8.20.1 Detailed Description

template<typename T> class Matrix3D< T >

Matrix 3D class.

Matrixes and operations.

Definition at line 147 of file Matrix.h.

8.20.2 Constructor & Destructor Documentation

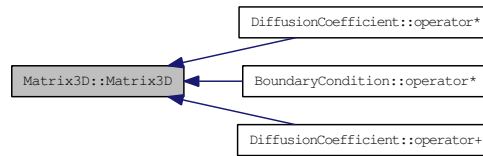
8.20.2.1 **template<typename T> Matrix3D< T >::Matrix3D ()** [inline]

Default constructor. Do nothing.

Definition at line 165 of file Matrix.h.

Referenced by [DiffusionCoefficient::operator*\(\)](#), [BoundaryCondition::operator*\(\)](#), and [DiffusionCoefficient::operator+\(\)](#).

Here is the caller graph for this function:



8.20.2.2 `template<class T> Matrix3D< T >::Matrix3D (const Matrix3D< T > & M)` [inline]

Constructor.

Create matrix equal to Matrix M.

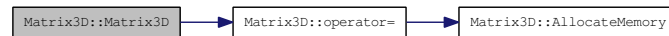
Parameters:

const Matrix3D<T> &M - Matrix M.

Definition at line 782 of file Matrix.cpp.

References Matrix3D< T >::initialized, and Matrix3D< T >::operator=().

Here is the call graph for this function:



8.20.2.3 `template<class T> Matrix3D< T >::Matrix3D (int x_size, int y_size, int z_size)` [inline]

Constructor.

Allocate memory.

Definition at line 769 of file Matrix.cpp.

References Matrix3D< T >::AllocateMemory(), and Matrix3D< T >::initialized.

Here is the call graph for this function:



8.20.2.4 `template<class T> Matrix3D< T >::~~Matrix3D ()` [inline]

Destructor.

Definition at line 791 of file Matrix.cpp.

References Matrix3D< T >::initialized, Matrix3D< T >::size_x, and Matrix3D< T >::size_y.

8.20.3 Member Function Documentation

8.20.3.1 `template<class T> Matrix3D< T > Matrix3D< T >::abs () [inline]`

Return absolute value of the matrix.

Definition at line 1185 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.2 `template<class T> void Matrix3D< T >::AllocateMemory (int x_size, int y_size, int z_size) [inline]`

Allocating memory and filling it with zero-values.

Todo

Do not fill initialized matrix with zeros.

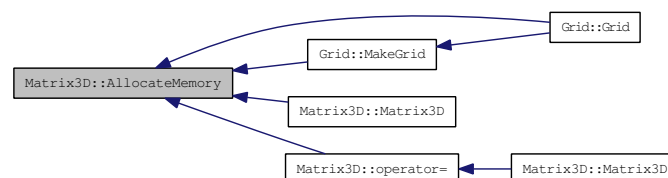
Reimplemented in [DiffusionCoefficient](#), and [GridElement](#).

Definition at line 802 of file Matrix.cpp.

References `i`, `Matrix3D< T >::initialized`, `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Referenced by `Grid::Grid()`, `Grid::MakeGrid()`, `Matrix3D< T >::Matrix3D()`, and `Matrix3D< T >::operator=()`.

Here is the caller graph for this function:



8.20.3.3 `template<class T> int Matrix3D< T >::index1d (int x, int y, int z) [inline]`

Returns corresponding index of 1d array.

Definition at line 820 of file Matrix.cpp.

References `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.4 `template<class T> T Matrix3D< T >::max () [inline]`

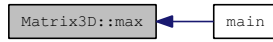
Return maximum value of the matrix.

Definition at line 1151 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Referenced by `main()`.

Here is the caller graph for this function:



8.20.3.5 `template<class T> T Matrix3D< T >::maxabs ()` [inline]

Return absolute maximum value of the matrix.

Definition at line 1168 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.6 `template<class T> T & Matrix3D< T >::operator() (int x, int y, int z)` [inline]

Return the (x,y,z) value of matrix.

Operator (x, y, z), returns value of element [x][y][z].

If `DEBUG_MODE` defined, check if matrix has been initialized.

Definition at line 857 of file Matrix.cpp.

References `Matrix3D< T >::initialized`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Referenced by `Matrix3D< double >::Value()`.

Here is the caller graph for this function:



8.20.3.7 `template<class T> Matrix3D< T > Matrix3D< T >::operator* (Matrix3D< T > & M)` [inline]

Multiply each element of the matrix to corresponds element of matrix M.

Reimplemented in [DiffusionCoefficient](#).

Definition at line 979 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.8 `template<class T> Matrix3D< T > Matrix3D< T >::operator* (T Val)` [inline]

Multiply eqach element of the matrix to Val.

Reimplemented in [DiffusionCoefficient](#), and [BoundaryCondition](#).

Definition at line 951 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.9 `template<class T> Matrix3D< T > Matrix3D< T >::operator+ (Matrix3D< T > & M)` [inline]

Add each element of the matrix to corresponds element of matrix M.

Reimplemented in [DiffusionCoefficient](#).

Definition at line 993 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.10 `template<class T> Matrix3D< T > Matrix3D< T >::operator- (Matrix3D< T > & M)` [inline]

Subtract each element of the matrix to corresponds element of matrix M.

Definition at line 1007 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.11 `template<class T> Matrix3D< T > Matrix3D< T >::operator/ (Matrix3D< T > & M)` [inline]

Divide each element of the matrix to corresponds element of matrix M.

Definition at line 965 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.12 `template<class T> Matrix3D< T > Matrix3D< T >::operator/ (T Val)` [inline]

Divide each element of the matrix to Val.

Definition at line 937 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.13 `template<class T> Matrix3D< T > & Matrix3D< T >::operator= (T Val)` [inline]

Makes Matrix equal to value Val.

Reimplemented in [DiffusionCoefficient](#), [DiffusionCoefficientsGroup](#), [BoundaryCondition](#), and [GridElement](#).

Definition at line 925 of file Matrix.cpp.

References `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.14 `template<class T> Matrix3D< T > & Matrix3D< T >::operator= (Matrix2D< T > & M)` [inline]

Makes 3D matrix from 2D matrix.

The 3rd dimension makes equal to 1.

Definition at line 903 of file Matrix.cpp.

References `Matrix3D< T >::AllocateMemory()`, `Matrix3D< T >::initialized`, `Matrix2D< T >::initialized`, `Matrix2D< T >::size_x`, `Matrix3D< T >::size_x`, `Matrix2D< T >::size_y`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Here is the call graph for this function:



8.20.3.15 `template<class T> Matrix3D< T > & Matrix3D< T >::operator= (const Matrix3D< T > & M) [inline]`

Makes matrix equal to Matrix M.

Parameters:

const `Matrix3D<T> &M` - Matrix M.

Reimplemented in [DiffusionCoefficient](#), [DiffusionCoefficientsGroup](#), and [GridElement](#).

Definition at line 873 of file `Matrix.cpp`.

References `Matrix3D< T >::AllocateMemory()`, `Matrix3D< T >::initialized`, `Matrix3D< T >::name`, `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Referenced by `Matrix3D< T >::Matrix3D()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.20.3.16 `template<class T> T ** Matrix3D< T >::operator[] (int i) [inline]`

Return the *i*-th pointer to 2d-array. Next `[j][k]` can be applied, so we have regular `[i][j][k]`.

Operator `[i]`, returns pointer to 2D array.

Next `[j][k]` can be applied to return value. If `DEBUG_MODE` defined, check if matrix has been initialized.

Parameters:

i - number of element to return

Definition at line 831 of file `Matrix.cpp`.

References `Matrix3D< T >::initialized`.

8.20.3.17 `template<class T> void Matrix3D< T >::readFromFile (string filename, Matrix3D< T > & grid_x, Matrix3D< T > & grid_y, Matrix3D< T > & grid_z) [inline]`

Read matrix data from file with grid.

Definition at line 1102 of file Matrix.cpp.

References `Matrix3D< T >::initialized`, `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.18 `template<class T> void Matrix3D< T >::readFromFile (string filename) [inline]`

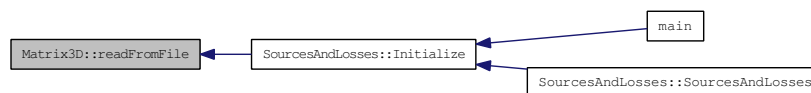
Read matrix data from file.

Definition at line 1064 of file Matrix.cpp.

References `Matrix3D< T >::initialized`, `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Referenced by `SourcesAndLosses::Initialize()`.

Here is the caller graph for this function:



8.20.3.19 `template<typename T> T& Matrix3D< T >::Value (int x, int y, int z) [inline]`

Definition at line 186 of file Matrix.h.

8.20.3.20 `template<class T> void Matrix3D< T >::writeToFile (string filename, Matrix3D< T > & grid_x, Matrix3D< T > & grid_y, Matrix3D< T > & grid_z) [inline]`

Write matrix to file, using 3 other matrixes as a grid (simply - write all 4 matrixes to the file).

File has two header lines.

Definition at line 1042 of file Matrix.cpp.

References `Matrix3D< T >::name`, `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

8.20.3.21 `template<class T> void Matrix3D< T >::writeToFile (string filename) [inline]`

Write matrix to file.

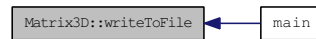
File has two header lines.

Definition at line 1022 of file Matrix.cpp.

References `Matrix3D< T >::name`, `Matrix3D< T >::size_x`, `Matrix3D< T >::size_y`, and `Matrix3D< T >::size_z`.

Referenced by `main()`.

Here is the caller graph for this function:



8.20.3.22 `template<class T> Matrix2D< T> Matrix3D< T>::xSlice(int p_x)` [inline]

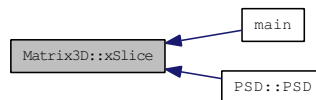
Make x-slice of 3d matrix - 2d matrix.

Definition at line 1203 of file Matrix.cpp.

References `Matrix3D< T>::name`, `Matrix2D< T>::name`, `Matrix3D< T>::size_y`, and `Matrix3D< T>::size_z`.

Referenced by `main()`, and `PSD::PSD()`.

Here is the caller graph for this function:



8.20.3.23 `template<class T> Matrix2D< T> Matrix3D< T>::ySlice(int p_y)` [inline]

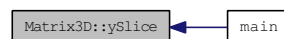
Make y-slice of 3d matrix - 2d matrix.

Definition at line 1219 of file Matrix.cpp.

References `Matrix3D< T>::name`, `Matrix2D< T>::name`, `Matrix3D< T>::size_x`, and `Matrix3D< T>::size_z`.

Referenced by `main()`.

Here is the caller graph for this function:



8.20.3.24 `template<class T> Matrix2D< T> Matrix3D< T>::zSlice(int p_z)` [inline]

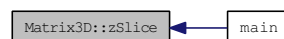
Make z-slice of 3d matrix - 2d matrix.

Definition at line 1235 of file Matrix.cpp.

References `Matrix3D< T>::name`, `Matrix2D< T>::name`, `Matrix3D< T>::size_x`, and `Matrix3D< T>::size_y`.

Referenced by `main()`.

Here is the caller graph for this function:



8.20.4 Member Data Documentation

8.20.4.1 `template<typename T> string Matrix3D< T >::change_ind`

Variables useful for changes tracking.

Definition at line 156 of file Matrix.h.

8.20.4.2 `template<typename T> bool Matrix3D< T >::initialized`

Flag, equal true if initialized.

Definition at line 155 of file Matrix.h.

Referenced by `Matrix3D< T >::AllocateMemory()`, `Matrix3D< double >::Matrix3D()`, `Matrix3D< T >::Matrix3D()`, `Matrix3D< T >::operator()`, `Matrix3D< T >::operator=()`, `Matrix3D< T >::operator[]()`, `Matrix3D< T >::readFromFile()`, and `Matrix3D< T >::~~Matrix3D()`.

8.20.4.3 `template<typename T> string Matrix3D< T >::name`

name of the Matrix

Definition at line 158 of file Matrix.h.

Referenced by `main()`, `Matrix3D< T >::operator=()`, `Matrix3D< T >::writeToFile()`, `Matrix3D< T >::xSlice()`, `Matrix3D< T >::ySlice()`, and `Matrix3D< T >::zSlice()`.

8.20.4.4 `template<typename T> int Matrix3D< T >::size_x`

Definition at line 157 of file Matrix.h.

Referenced by `Matrix3D< T >::abs()`, `Matrix3D< T >::AllocateMemory()`, `Matrix3D< T >::max()`, `Matrix3D< T >::maxabs()`, `Matrix3D< T >::operator*()`, `Matrix3D< T >::operator+()`, `Matrix3D< T >::operator-()`, `Matrix3D< T >::operator/()`, `Matrix3D< T >::operator=()`, `Matrix3D< T >::readFromFile()`, `Matrix3D< T >::writeToFile()`, `Matrix3D< T >::ySlice()`, `Matrix3D< T >::zSlice()`, and `Matrix3D< T >::~~Matrix3D()`.

8.20.4.5 `template<typename T> int Matrix3D< T >::size_y`

Definition at line 157 of file Matrix.h.

Referenced by `Matrix3D< T >::abs()`, `Matrix3D< T >::AllocateMemory()`, `Matrix3D< T >::index1d()`, `Matrix3D< T >::max()`, `Matrix3D< T >::maxabs()`, `Matrix3D< T >::operator()`, `Matrix3D< T >::operator*()`, `Matrix3D< T >::operator+()`, `Matrix3D< T >::operator-()`, `Matrix3D< T >::operator/()`, `Matrix3D< T >::operator=()`, `Matrix3D< T >::readFromFile()`, `Matrix3D< T >::writeToFile()`, `Matrix3D< T >::xSlice()`, `Matrix3D< T >::zSlice()`, and `Matrix3D< T >::~~Matrix3D()`.

8.20.4.6 `template<typename T> int Matrix3D< T >::size_z`

size x, size y, size z

Definition at line 157 of file Matrix.h.

Referenced by `Matrix3D< T >::abs()`, `Matrix3D< T >::AllocateMemory()`, `Matrix3D< T >::index1d()`, `Matrix3D< T >::max()`, `Matrix3D< T >::maxabs()`, `Matrix3D< T >::operator()`, `Matrix3D< T >::operator=()`, `Matrix3D< T >::readFromFile()`, `Matrix3D< T >::writeToFile()`, `Matrix3D< T >::xSlice()`, `Matrix3D< T >::zSlice()`, and `Matrix3D< T >::~~Matrix3D()`.

`>::operator*()`, `Matrix3D< T >::operator+()`, `Matrix3D< T >::operator-()`, `Matrix3D< T >::operator/()`, `Matrix3D< T >::operator=()`, `Matrix3D< T >::readFromFile()`, `Matrix3D< T >::writeToFile()`, `Matrix3D< T >::xSlice()`, and `Matrix3D< T >::ySlice()`.

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

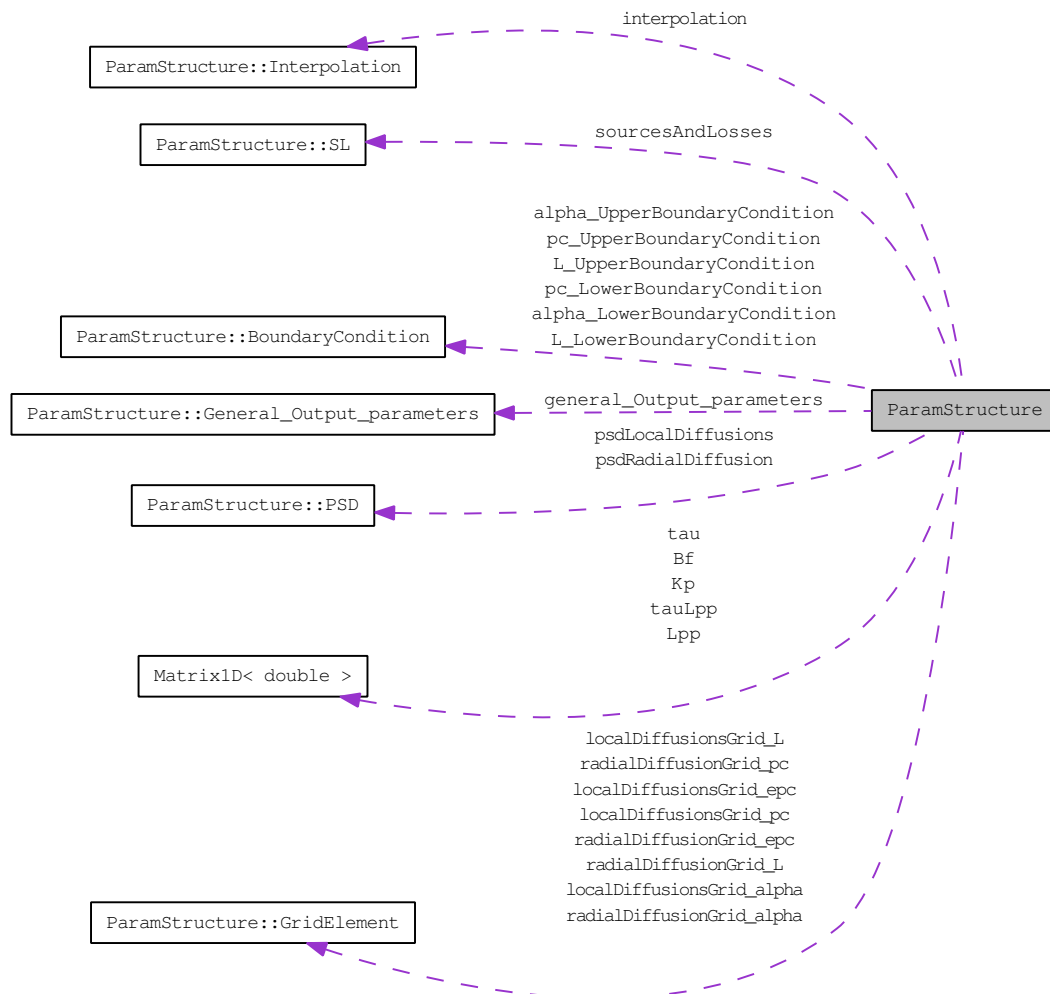
- [Matrix.h](#)
- [Matrix.cpp](#)

8.21 ParamStructure Struct Reference

Main parameters structure.

```
#include <Parameters.h>
```

Collaboration diagram for ParamStructure:



Classes

- struct [BoundaryCondition](#)
Boundary conditions parameters structure.
- struct [General_Output_parameters](#)
General programm output parameters structure.
- struct [GridElement](#)
Grid element parameters structure.

- struct [Interpolation](#)
Interpolation parameters structure
- struct [PSD](#)
PSD parameters structure.
- struct [SL](#)
Sources and losses.

Public Member Functions

- bool [Load_parameters](#) (string filename)
Loads parameters from file.

Public Attributes

- int [outputLvl](#)
Detalization level of screen output.
- double [nDays](#)
Simulation total time, days.
- double [timeStep](#)
Time step, hours.
- int [totalIterationsNumber](#)
Total number of iterations.
- bool [useRadialDifusion](#)
Using diffusions flags.
- bool [useAlphaDifusion](#)
Using diffusions flags.
- bool [useEnergyDifusion](#)
Using diffusions flags.
- bool [useEnergyAlphaMixedTerms](#)
Using mixed terms flags. DLL type (which method to use to calculate). Check StrToVal(string input, DLL-Types &place) for known values.
- string [DLLType](#)
- [Matrix1D](#)< double > [Kp](#)
Kp array.
- string [useKp](#)
Flag, using Kp index. constant - use constant value, file - read from file.

- double [constKp](#)
constant Kp value (if it's constant).
- string [fileKp](#)
Kp file name.
- [Matrix1D< double > Bf](#)
Boundary flux array.
- string [useBf](#)
Flag, using boundary flux dependences if equal true.
- double [constBf](#)
constant Bf value (if it's constant).
- string [fileBf](#)
Bf file name.
- [Matrix1D< double > Lpp](#)
Lpp array.
- string [useLpp](#)
Flag, using plasma pause location dependences if equal true.
- double [constLpp](#)
constant Lpp value (if it's constant).
- string [fileLpp](#)
Lpp file name.
- [Matrix1D< double > tau](#)
Lifetime out of the plasmasphere.
- [Matrix1D< double > tauLpp](#)
Lifetime inside of the plasmasphere.
- bool [outputModelMatrix](#)
Output model matrix.
- bool [NoNegative](#)
Artificially no negative PSD.
- struct [ParamStructure::General_Output_parameters](#) [general_Output_parameters](#)
General programm output parameters structure.
- string [radialDiffusionGrid_type](#)
Local diffusions grid type.
- string [localDiffusionsGrid_type](#)

Radial diffusion grid type.

- string `radialDiffusionGrid_filename`
filename, if grid load from file
- string `localDiffusionsGrid_filename`
filename, if grid load from file
- struct `ParamStructure::GridElement radialDiffusionGrid_L`
Grid element parameters structure.
- struct `ParamStructure::GridElement radialDiffusionGrid_pc`
- struct `ParamStructure::GridElement radialDiffusionGrid_alpha`
- struct `ParamStructure::GridElement radialDiffusionGrid_epc`
- struct `ParamStructure::GridElement localDiffusionsGrid_L`
- struct `ParamStructure::GridElement localDiffusionsGrid_pc`
- struct `ParamStructure::GridElement localDiffusionsGrid_alpha`
- struct `ParamStructure::GridElement localDiffusionsGrid_epc`
- struct `ParamStructure::BoundaryCondition L_LowerBoundaryCondition`
Boundary conditions parameters structure.
- struct `ParamStructure::BoundaryCondition L_UpperBoundaryCondition`
- struct `ParamStructure::BoundaryCondition pc_LowerBoundaryCondition`
- struct `ParamStructure::BoundaryCondition pc_UpperBoundaryCondition`
- struct `ParamStructure::BoundaryCondition alpha_LowerBoundaryCondition`
- struct `ParamStructure::BoundaryCondition alpha_UpperBoundaryCondition`
- struct `ParamStructure::PSD psdRadialDiffusion`
PSD parameters structure.
- struct `ParamStructure::PSD psdLocalDiffusions`
PSD parameters for local diffusions.
- struct `ParamStructure::SL sourcesAndLosses`
Sources and losses.
- struct `ParamStructure::Interpolation interpolation`
Interpolation parameters structure
- `DiffusionCoefficientParamStructureList DxxParamStructureList`
List of diffusion coefficients parameters.

8.21.1 Detailed Description

Main parameters structure.

The structure was modified without multiple substructures according to request of project PI

Definition at line 84 of file Parameters.h.

8.21.2 Member Function Documentation

8.21.2.1 bool ParamStructure::Load_parameters (string filename)

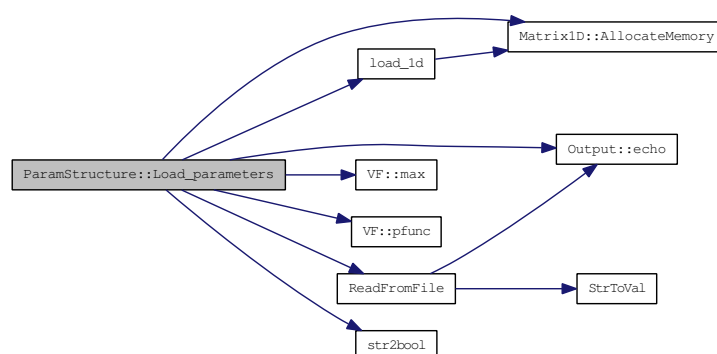
Loads parameters from file.

Definition at line 71 of file Parameters.cpp.

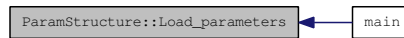
References `Matrix1D< T >::AllocateMemory()`, `alpha_LowerBoundaryCondition`, `alpha_UpperBoundaryCondition`, `ParamStructure::PSD::approximationMethod`, `Bf`, `constBf`, `constKp`, `constLpp`, `DLLType`, `DxxParamStructureList`, `Output::echo()`, `fileBf`, `fileKp`, `fileLpp`, `ParamStructure::BoundaryCondition::filename`, `ParamStructure::General_Output_parameters::fileName1D`, `ParamStructure::General_Output_parameters::folderName`, `general_Output_parameters`, `i`, `ParamStructure::PSD::initial_PSD_filename`, `ParamStructure::PSD::initial_PSD_inner_psd`, `ParamStructure::PSD::initial_PSD_J_L7_function`, `ParamStructure::PSD::initial_PSD_Kp0`, `ParamStructure::PSD::initial_PSD_outer_psd`, `ParamStructure::PSD::initial_PSD_some_constant_value`, `ParamStructure::PSD::initial_PSD_tauSteadyState`, `ParamStructure::PSD::initial_PSD_Type`, `interpolation`, `ParamStructure::General_Output_parameters::iterStep`, `Kp`, `L_LowerBoundaryCondition`, `L_UpperBoundaryCondition`, `ParamStructure::Interpolation::linearSplineCoef`, `load_1d()`, `localDiffusionsGrid_alpha`, `localDiffusionsGrid_epc`, `localDiffusionsGrid_filename`, `localDiffusionsGrid_L`, `localDiffusionsGrid_pc`, `localDiffusionsGrid_type`, `ParamStructure::General_Output_parameters::logFileName`, `Lpp`, `ParamStructure::GridElement::max`, `VF::max()`, `ParamStructure::Interpolation::maxSecondDerivative`, `ParamStructure::GridElement::min`, `ParamStructure::GridElement::name`, `nDays`, `NoNegative`, `ParamStructure::PSD::output_PSD_filename4D`, `ParamStructure::PSD::output_PSD_folderName`, `ParamStructure::PSD::output_PSD_timeStep`, `outputLvl`, `outputModelMatrix`, `pc_LowerBoundaryCondition`, `pc_UpperBoundaryCondition`, `VF::pfunc()`, `psdLocalDiffusions`, `psdRadialDiffusion`, `radialDiffusionGrid_alpha`, `radialDiffusionGrid_epc`, `radialDiffusionGrid_filename`, `radialDiffusionGrid_L`, `radialDiffusionGrid_pc`, `radialDiffusionGrid_type`, `ReadFromFile()`, `ParamStructure::GridElement::size`, `ParamStructure::SL::SL_E_min`, `ParamStructure::SL::SL_E_min_filename`, `ParamStructure::SL::SL_L_top`, `ParamStructure::SL::SL_L_top_filename`, `ParamStructure::PSD::SOL_i_max`, `ParamStructure::PSD::SOL_max_iter_err`, `ParamStructure::PSD::SOL_maxiter`, `ParamStructure::PSD::solutionMethod`, `sourcesAndLosses`, `str2bool()`, `tau`, `tauLpp`, `ParamStructure::General_Output_parameters::timeStep`, `timeStep`, `totalIterationsNumber`, `ParamStructure::Interpolation::type`, `ParamStructure::BoundaryCondition::type`, `useAlphaDifusion`, `useBf`, `useEnergyAlphaMixedTerms`, `useEnergyDifusion`, `useKp`, `ParamStructure::Interpolation::useLog`, `ParamStructure::GridElement::useLogScale`, `useLpp`, `useRadialDifusion`, and `ParamStructure::BoundaryCondition::value`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



8.21.3 Member Data Documentation

8.21.3.1 struct ParamStructure::BoundaryCondition ParamStructure::alpha_LowerBoundaryCondition

Referenced by `Load_parameters()`, and `main()`.

8.21.3.2 struct ParamStructure::BoundaryCondition ParamStructure::alpha_UpperBoundaryCondition

Referenced by `Load_parameters()`, and `main()`.

8.21.3.3 Matrix1D<double> ParamStructure::Bf

Boundary flux array.

Definition at line 100 of file `Parameters.h`.

Referenced by `Load_parameters()`, and `main()`.

8.21.3.4 double ParamStructure::constBf

constant Bf value (if it's constant).

Definition at line 102 of file `Parameters.h`.

Referenced by `Load_parameters()`.

8.21.3.5 double ParamStructure::constKp

constant Kp value (if it's constant).

Definition at line 98 of file `Parameters.h`.

Referenced by `Load_parameters()`.

8.21.3.6 double ParamStructure::constLpp

constant Lpp value (if it's constant).

Definition at line 106 of file `Parameters.h`.

Referenced by `Load_parameters()`.

8.21.3.7 string ParamStructure::DLLType

Definition at line 95 of file `Parameters.h`.

Referenced by Load_parameters(), and main().

8.21.3.8 DiffusionCoefficientParamStructureList ParamStructure::DxxParamStructureList

List of diffusion coefficients parameters.

Definition at line 225 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.9 string ParamStructure::fileBf

Bf file name.

Definition at line 103 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.10 string ParamStructure::fileKp

Kp file name.

Definition at line 99 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.11 string ParamStructure::fileLpp

Lpp file name.

Definition at line 107 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.12 struct ParamStructure::General_Output_parameters ParamStructure::general_Output_parameters

General programm output parameters structure.

Referenced by Load_parameters(), and main().

8.21.3.13 struct ParamStructure::Interpolation ParamStructure::interpolation

Interpolation parameters structure

Interpolation parameters.

Referenced by Load_parameters(), and main().

8.21.3.14 Matrix1D<double> ParamStructure::Kp

Kp array.

Definition at line 96 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.15 struct ParamStructure::BoundaryCondition ParamStructure::L_LowerBoundaryCondition

Boundary conditions parameters structure.

Referenced by Load_parameters(), and main().

8.21.3.16 struct ParamStructure::BoundaryCondition ParamStructure::L_UpperBoundaryCondition

Referenced by Load_parameters(), and main().

8.21.3.17 struct ParamStructure::GridElement ParamStructure::localDiffusionsGrid_alpha

Referenced by Load_parameters(), and main().

8.21.3.18 struct ParamStructure::GridElement ParamStructure::localDiffusionsGrid_epc

Referenced by Load_parameters(), and main().

8.21.3.19 string ParamStructure::localDiffusionsGrid_filename

filename, if grid load from file

Definition at line 129 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.20 struct ParamStructure::GridElement ParamStructure::localDiffusionsGrid_L

Referenced by Load_parameters(), and main().

8.21.3.21 struct ParamStructure::GridElement ParamStructure::localDiffusionsGrid_pc

Referenced by Load_parameters(), and main().

8.21.3.22 string ParamStructure::localDiffusionsGrid_type

Radial diffusion grid type.

Definition at line 125 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.23 Matrix1D<double> ParamStructure::Lpp

Lpp array.

Definition at line 104 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.24 double ParamStructure::nDays

Simulation total time, days.

Definition at line 87 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.25 bool ParamStructure::NoNegative

Artificially no negative [PSD](#).

Definition at line 112 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.26 int ParamStructure::outputLvl

Detalization level of screen output.

Definition at line 86 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.27 bool ParamStructure::outputModelMatrix

[Output](#) model matrix.

Definition at line 111 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.28 struct ParamStructure::BoundaryCondition ParamStructure::pc_LowerBoundaryCondition

Referenced by Load_parameters(), and main().

8.21.3.29 struct ParamStructure::BoundaryCondition ParamStructure::pc_UpperBoundaryCondition

Referenced by Load_parameters(), and main().

8.21.3.30 struct ParamStructure::PSD ParamStructure::psdLocalDiffusions

PSD parameters for local diffusions.

Referenced by Load_parameters(), and main().

8.21.3.31 struct ParamStructure::PSD ParamStructure::psdRadialDiffusion

PSD parameters structure.

PSD parameters for radial diffusion.

Referenced by Load_parameters(), and main().

8.21.3.32 struct ParamStructure::GridElement ParamStructure::radialDiffusionGrid_alpha

Referenced by Load_parameters(), and main().

8.21.3.33 struct ParamStructure::GridElement ParamStructure::radialDiffusionGrid_epc

Referenced by Load_parameters(), and main().

8.21.3.34 string ParamStructure::radialDiffusionGrid_filename

filename, if grid load from file

Definition at line 128 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.35 struct ParamStructure::GridElement ParamStructure::radialDiffusionGrid_L

Grid element parameters structure.

Referenced by Load_parameters(), and main().

8.21.3.36 struct ParamStructure::GridElement ParamStructure::radialDiffusionGrid_pc

Referenced by Load_parameters(), and main().

8.21.3.37 string ParamStructure::radialDiffusionGrid_type

Local diffusions grid type.

Definition at line 125 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.38 struct ParamStructure::SL ParamStructure::sourcesAndLosses

Sources and losses.

Referenced by Load_parameters(), and main().

8.21.3.39 Matrix1D<double> ParamStructure::tau

Lifetime out of the plasmasphere.

Definition at line 108 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.40 Matrix1D<double> ParamStructure::tauLpp

Lifetime inside of the plasmasphere.

Definition at line 109 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.41 double ParamStructure::timeStep

Time step, hours.

Definition at line 88 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.42 int ParamStructure::totalIterationsNumber

Total number of iterations.

Definition at line 89 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.43 bool ParamStructure::useAlphaDifusion

Using diffusions flags.

Definition at line 91 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.44 string ParamStructure::useBf

Flag, using boundary flux dependences if equal true.

Definition at line 101 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.45 bool ParamStructure::useEnergyAlphaMixedTerms

Using mixed terms flags. DLL type (which method to use to calculate). Check StrToVal(string input, DLLTypes &place) for known values.

Definition at line 93 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.46 bool ParamStructure::useEnergyDifusion

Using diffusions flags.

Definition at line 92 of file Parameters.h.

Referenced by Load_parameters(), and main().

8.21.3.47 string ParamStructure::useKp

Flag, using Kp index. constant - use constant value, file - read from file.

Definition at line 97 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.48 string ParamStructure::useLpp

Flag, using plasma pause location dependences if equal true.

Definition at line 105 of file Parameters.h.

Referenced by Load_parameters().

8.21.3.49 bool ParamStructure::useRadialDifusion

Using diffusions flags.

Definition at line 90 of file Parameters.h.

Referenced by Load_parameters(), and main().

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

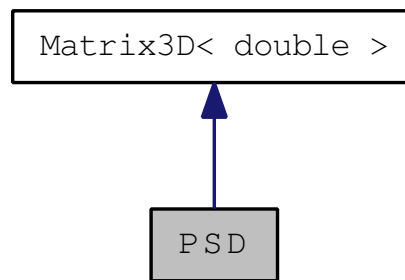
- [Parameters.h](#)
- [Parameters.cpp](#)

8.22 PSD Class Reference

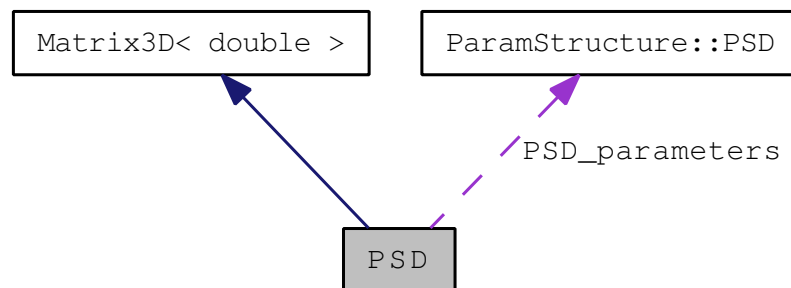
Phase Space Density class.

```
#include <PSD.h>
```

Inheritance diagram for PSD:



Collaboration diagram for PSD:



Public Member Functions

- `PSD ()`
Matrix for split method.
- `PSD (ParamStructure::PSD parameters, Grid &grid)`
Constructor.
- `PSD (ParamStructure::PSD parameters, Grid &grid, PSD &otherPSD, Grid &otherPSD_grid, ParamStructure::Interpolation constantsInterpolation)`
- `PSD (ParamStructure::PSD parameters, Grid &grid, BoundaryCondition L_-, UpperBoundaryCondition)`
Constructor.
- `~PSD ()`
Destructor.
- `void Initialize (ParamStructure::PSD parameters, Grid &grid, Matrix2D< double > L_-, UpperBoundaryCondition=Matrix2D< double >())`

Initializing: storing parameters, loading initial values, making boundary conditions, initializing output parameters Simply, it is a creation of the object.

- void [LoadInitialValue](#) ([ParamStructure::PSD](#) parameters, [Grid](#) &grid, [Matrix2D](#)< double > L_UpperBoundaryCondition=[Matrix2D](#)< double >())

Loading initial values - from a file or other sources.

- void [Interpolate](#) ([PSD](#) &otherPSD, [ParamStructure::Interpolation](#) interpolationParameters, [Grid](#) &oldGrid, [Grid](#) &newGrid, [Matrix2D](#)< double > newGrid_pc_lowerBoundaryCondition, [Matrix2D](#)< double > newGrid_pc_upperBoundaryCondition)

Interpolation function.

- void [Load_initial_f](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double tau, double Kp, double min_psd=1.e-99, string J_L7_function="J_L7", double fb_out=1, double fb_in=0)

Calculate initial PSD from steady state.

- void [Load_initial_f](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, double tau, double Kp, [Matrix2D](#)< double > L_UpperBoundaryCondition, double min_psd=1.e-99, double fb_out=1, double fb_in=0)

Calculate initial PSD from steady state using boundary conditions.

- void [Load_initial_f_2d](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, const char *filename)

Load initial PSD from 2d-file (for 2d calculations).

- void [Load_initial_f_file](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, const char *filename, bool with_grid)

Load initial PSD from file.

- void [DiffusionMixTermExplicit](#) (double dt, double Lpp, [DiffusionCoefficient](#) &Dpca, [DiffusionCoefficient](#) &DpcaLpp, [GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, [Matrix3D](#)< double > Jacobian, [Matrix2D](#)< double > pc_lowerBoundaryCondition, [Matrix2D](#)< double > pc_upperBoundaryCondition, [Matrix2D](#)< double > alpha_lowerBoundaryCondition, [Matrix2D](#)< double > alpha_upperBoundaryCondition, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType)

Mixed terms calculation by explicit method.

- void [Diffusion_alpha](#) (double dt, double Lpp, [DiffusionCoefficient](#) &Daa, [DiffusionCoefficient](#) &DaaLpp, [GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, [Matrix3D](#)< double > Jacobian, [Matrix2D](#)< double > alpha_lowerBoundaryCondition, [Matrix2D](#)< double > alpha_upperBoundaryCondition, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType)

Pitch angle diffusion calculation function.

- void [Diffusion_pc](#) (double dt, double Lpp, [DiffusionCoefficient](#) &Dpcpc, [DiffusionCoefficient](#) &DpcpcLpp, [GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, [Matrix3D](#)< double > Jacobian, [Matrix2D](#)< double > pc_lowerBoundaryCondition, [Matrix2D](#)< double > pc_upperBoundaryCondition, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType)

Energy diffusion calculation function.

- void [Diffusion_L](#) (double dt, double Lpp, [DiffusionCoefficient](#) &DLL, [GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, [Matrix3D](#)< double > Jacobian, [Matrix2D](#)< double > lowerBoundaryCondition, [Matrix2D](#)< double > upperBoundaryCondition, string lowerBoundaryCondition_calculationType, string upperBoundaryCondition_calculationType, double tau, double tauLpp)

Radial diffusion calculation function.

- void [Diffusion_pc_alpha](#) (double dt, double Lpp, [DiffusionCoefficient](#) &Dpcpc, [DiffusionCoefficient](#) &DpcpcLpp, [DiffusionCoefficient](#) &Daa, [DiffusionCoefficient](#) &DaaLpp, [DiffusionCoefficient](#) &Dpca, [DiffusionCoefficient](#) &DpcaLpp, [GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, [Matrix3D](#)< double > Jacobian, [Matrix2D](#)< double > pc_lowerBoundaryCondition, [Matrix2D](#)< double > pc_upperBoundaryCondition, [Matrix2D](#)< double > alpha_lowerBoundaryCondition, [Matrix2D](#)< double > alpha_upperBoundaryCondition, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType)

Radial diffusion calculation function.

- void [SourcesAndLosses](#) ([GridElement](#) &L, [GridElement](#) &pc, [GridElement](#) &alpha, [Matrix3D](#)< double > &SL, double dt, double Lpp, double tau, double tauLpp)

Sources and losses term from the FP eq.

- void [Output_without_grid](#) (double time)

PSD output.

Public Attributes

- [ParamStructure::PSD](#) [PSD_parameters](#)
- ofstream * [output_without_grid_file](#)

8.22.1 Detailed Description

Phase Space Density class.

Do diffusions and store the result in parent class [Matrix3D](#). Do all possible operations with PSD (like output, loading, etc).

Definition at line 33 of file PSD.h.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 [PSD::PSD\(\)](#) `[inline]`

Matrix for split method.

Matrix for radial diffusion. 2D matrix (for each pc and alpha) of 3-diagonal matrixes (for 1d radial diffusion problems). Default constructor - does nothing., mark initialized = false, means memory for arrays of the class was not allocated and the calculations, which should be done before using (if any) were not made.

Definition at line 47 of file PSD.h.

8.22.2.2 PSD::PSD (ParamStructure::PSD parameters, Grid & grid)

Constructor.

Creates grid, initialize parent class [Matrix3D](#) and run it's own Initialization.

We usually can use an empty constructor, like [PSD\(\)](#), and then do everything we need by functions, like [PSD::AllocateMemory\(\)](#), [PSD::Initialize\(\)](#) etc, but sometimes we have to completely define an object right in the moment of the creation (like in stacks) and also it is shorter to use one line of code instead of 3, so we need more complicated constructors. Sometimes we can NOT define an object in the moment of construction so, we have to use an empty constructor and define the rest by functions (if we don't know everything we need for that, but want to create an object).

So, to have both options, we do not write initialization of objects (construction of objects, whatever) in the constructors. The code has functions AllocateMemory, Initialize etc, which are called from constructors. And also can be called separately. We can create an abject by calling constructor with parameters, which call additional initialization functions, or create an object by calling empty constructor and then all initialization functions.

Parameters:

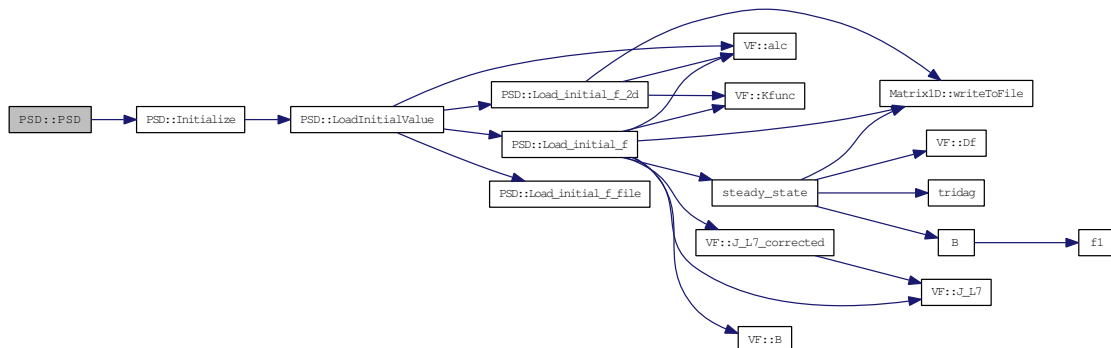
parameters - PSD parameters structure

&grid - grid, calculated somewhere

Definition at line 51 of file PSD.cpp.

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

Here is the call graph for this function:



8.22.2.3 PSD::PSD (ParamStructure::PSD parameters, Grid & grid, PSD & otherPSD, Grid & otherPSD_grid, ParamStructure::Interpolation constantsInterpolation)

8.22.2.4 PSD::PSD (ParamStructure::PSD parameters, Grid & grid, BoundaryCondition L_UpperBoundaryCondition)

Constructor.

Creates grid, initialize creates parent class [Matrix3D](#) and run it's own Initialization.

Parameters:

parameters - PSD parameters structure

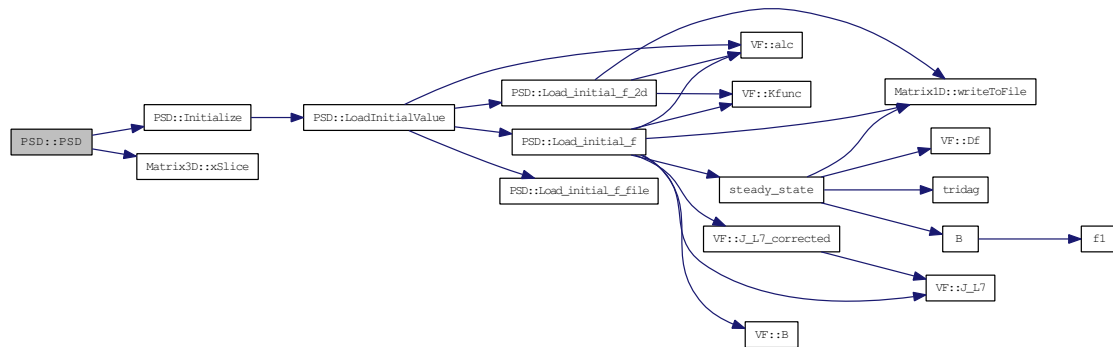
&grid - grid, calculated somewhere

&L_UpperBoundaryCondition - upper boundary for steady state calculation

Definition at line 62 of file PSD.cpp.

References ParamStructure::PSD::initial_PSD_Type, Initialize(), BoundaryCondition::initialType, and Matrix3D< T >::xSlice().

Here is the call graph for this function:



8.22.2.5 PSD::~~PSD ()

Destructor.

Definition at line 31 of file PSD.cpp.

8.22.3 Member Function Documentation

8.22.3.1 void PSD::Diffusion_alpha (double *dt*, double *Lpp*, DiffusionCoefficient & *Daa*, DiffusionCoefficient & *DaaLpp*, GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, Matrix3D< double > *Jacobian*, Matrix2D< double > *alpha_lowerBoundaryCondition*, Matrix2D< double > *alpha_upperBoundaryCondition*, string *alpha_lowerBoundaryCondition_calculationType*, string *alpha_upperBoundaryCondition_calculationType*)

Pitch angle diffusion calculation function.

Takes diffusion coefficients, boundary conditions and grid elements as input. It calculates model matrix by MakeModelMatrix_3D for 3D problem: $A_{3D} * PSD_{3D}(t+1) = B_{3D} * PSD_{3D}(t) + C_{3D}$ then splits it for 1D problems: $A * PSD_{1D}(t+1) = B * PSD_{1D}(t) + C$ and solve by tridiag method.

Parameters:

dt - time step

Lpp - plasma pause location

&Daa - L diffusion coefficient

&DaaLpp - L diffusion coefficient

&L - grid element L

&pc - grid elelent pc

&alpha - grid element alpha

&Jacobian - jacobian

&alpha_lowerBoundaryCondition - lower boundary condition

&alpha_upperBoundaryCondition - upper boundary condition

&alpha_lowerBoundaryCondition_calculationType - lower boundary condition type (on value/on derivative)

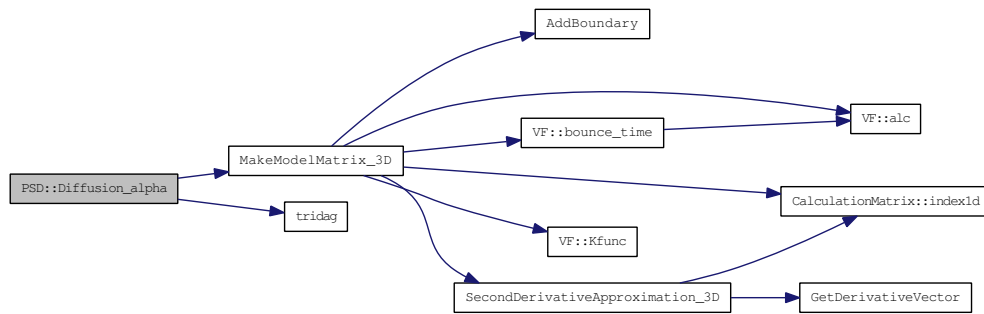
&alpha_upperBoundaryCondition_calculationType - upper boundary condition type (on value/on derivative)

Definition at line 314 of file PSD.cpp.

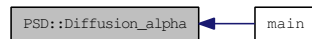
References MakeModelMatrix_3D(), GridElement::size, and tridag().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.2 void PSD::Diffusion_L (double dt, double Lpp, DiffusionCoefficient & DLL, GridElement & L, GridElement & pc, GridElement & alpha, Matrix3D< double > Jacobian, Matrix2D< double > L_lowerBoundaryCondition, Matrix2D< double > L_upperBoundaryCondition, string L_lowerBoundaryCondition_calculationType, string L_upperBoundaryCondition_calculationType, double tau, double tauLpp)

Radial diffusion calculation function.

Takes diffusion coefficients, boundary conditions and grid elements as parameters. It calculates model matrix by makeMatrix and then solve it by SolveMatrix method.

Parameters:

dt - time step

Lpp - plasma pause location

&DLL - L diffusion coefficient

&L - grid element L

&pc - grid element pc

&alpha - grid element alpha

&lowerBoundaryCondition - lower boundary condition on L

&upperBoundaryCondition - upper boundary condition on L

tau - life time upper location of the plasma pause

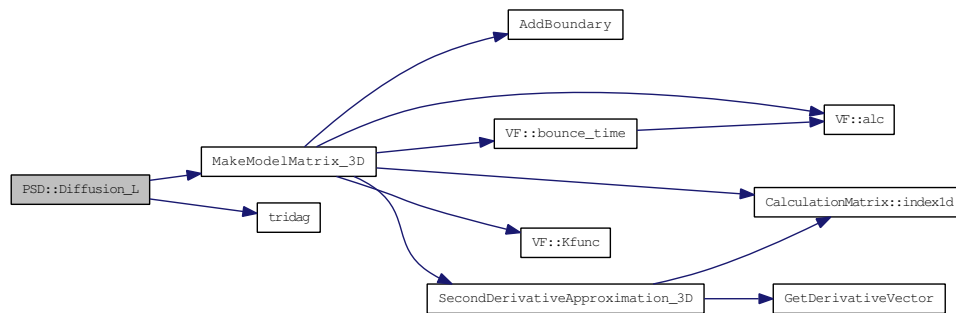
tauLpp - life time lower location of the plasma pause

Definition at line 582 of file PSD.cpp.

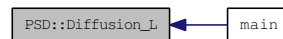
References MakeModelMatrix_3D(), GridElement::size, and tridag().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.3 void PSD::Diffusion_pc (double dt, double Lpp, DiffusionCoefficient & Dpcpc, DiffusionCoefficient & DpcpcLpp, GridElement & L, GridElement & pc, GridElement & alpha, Matrix3D< double > Jacobian, Matrix2D< double > pc_lowerBoundaryCondition, Matrix2D< double > pc_upperBoundaryCondition, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType)

Energy diffusion calculation function.

Takes diffusion coefficients, boundary conditions and grid elements as input. It calculates model matrix by MakeModelMatrix_3D for 3D problem: $A_{3D} * PSD_{3D}(t+1) = B_{3D} * PSD_{3D}(t) + C_{3D}$ then splits it for 1D problems: $A * PSD_{1D}(t+1) = B * PSD_{1D}(t) + C$ and solve by tridiag method.

Parameters:

dt - time step

Lpp - plasma pause location

&Dpcpc - L diffusion coefficient

&DpcpcLpp - L diffusion coefficient

&L - grid element L

&pc - grid elelent pc

&alpha - grid element alpha

&Jacobian - jacobian

&pc_lowerBoundaryCondition - lower boundary condition

&pc_upperBoundaryCondition - upper boundary condition

&pc_lowerBoundaryCondition_calculationType - lower boundary condition type (on value/on derivative)

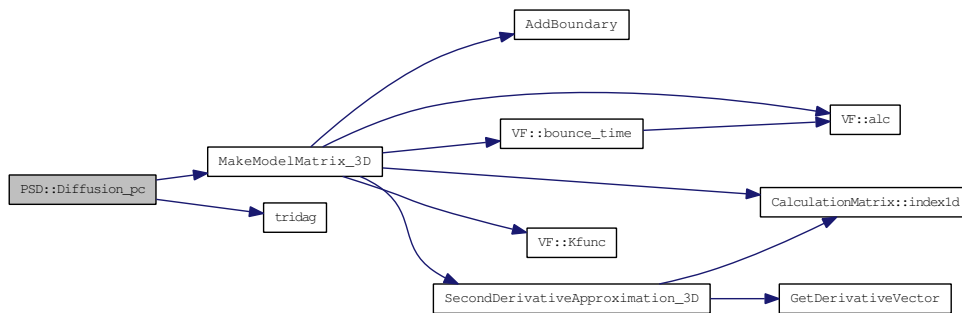
&pc_upperBoundaryCondition_calculationType - upper boundary condition type (on value/on derivative)

Definition at line 449 of file PSD.cpp.

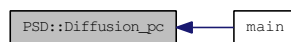
References MakeModelMatrix_3D(), GridElement::size, and tridag().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.4 void PSD::Diffusion_pc_alpha (double dt, double Lpp, DiffusionCoefficient & Dpcpc, DiffusionCoefficient & DpcpcLpp, DiffusionCoefficient & Daa, DiffusionCoefficient & DaaLpp, DiffusionCoefficient & Dpca, DiffusionCoefficient & DpcaLpp, GridElement & L, GridElement & pc, GridElement & alpha, Matrix3D< double > Jacobian, Matrix2D< double > pc_lowerBoundaryCondition, Matrix2D< double > pc_upperBoundaryCondition, Matrix2D< double > alpha_lowerBoundaryCondition, Matrix2D< double > alpha_upperBoundaryCondition, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType)

Radial diffusion calculation function.

Takes diffusion coefficients, boundary conditions and grid elements as parameters. It calculates model matrix by makeMatrix and then solve it by SolveMatrix method.

Parameters:

dt - time step

Lpp - plasma pause location

&Dpcpc - diffusion coefficient

&DpcpcLpp - diffusion coefficient

&Daa - diffusion coefficient

&DaaLpp - diffusion coefficient

&Dpca - diffusion coefficient

&DpcaLpp - diffusion coefficient

&L - grid element L

&pc - grid elelent pc

&alpha - grid element alpha

&pc_lowerBoundaryCondition - lower boundary condition

&pc_upperBoundaryCondition - upper boundary condition

&alpha_lowerBoundaryCondition - lower boundary condition

&alpha_upperBoundaryCondition - upper boundary condition

tau - life time upper location of the plasma pause

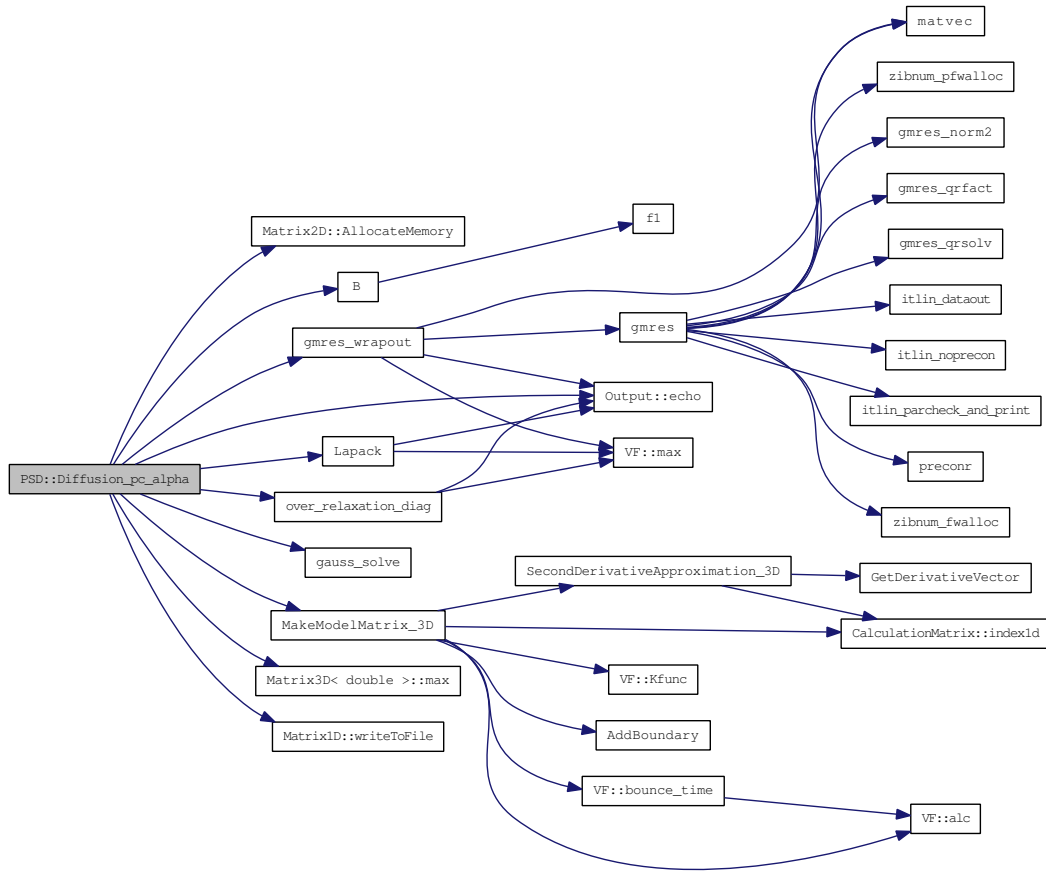
tauLpp - life time lower location of the plasma pause

Definition at line 723 of file PSD.cpp.

References Matrix2D< T >::AllocateMemory(), B(), Output::echo(), gauss_solve(), gmres_wrapout(), Matrix2D< T >::initialized, Lapack(), MakeModelMatrix_3D(), Matrix3D< double >::max(), over_relaxation_diag(), PSD_parameters, GridElement::size, ParamStructure::PSD::SOL_i_max, ParamStructure::PSD::SOL_max_iter_err, ParamStructure::PSD::SOL_maxiter, ParamStructure::PSD::solutionMethod, and Matrix1D< T >::writeToFile().

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.5 void PSD::DiffusionMixTermExplicit (double *dt*, double *Lpp*, DiffusionCoefficient & *Dpca*, DiffusionCoefficient & *DpcaLpp*, GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, Matrix3D< double > *Jacobian*, Matrix2D< double > *pc_lowerBoundaryCondition*, Matrix2D< double > *pc_upperBoundaryCondition*, Matrix2D< double > *alpha_lowerBoundaryCondition*, Matrix2D< double > *alpha_upperBoundaryCondition*, string *pc_lowerBoundaryCondition_calculationType*, string *pc_upperBoundaryCondition_calculationType*, string *alpha_lowerBoundaryCondition_calculationType*, string *alpha_upperBoundaryCondition_calculationType*)

Mixed terms calculation by explicit method.

Parameters:

dt - time step

Lpp - plasma pause location

&Dpca - pc-alpha diffusion coefficient

&DpcaLpp - pc-alpha diffusion coefficient under plasma pause location

&L - grid element L

&pc - grid element pc

&alpha - grid element alpha

&pc_lowerBoundaryCondition - lower boundary condition on pc

&pc_upperBoundaryCondition - upper boundary condition on pc

&alpha_lowerBoundaryCondition - lower boundary condition on alpha

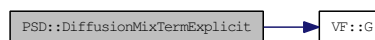
&alpha_upperBoundaryCondition - upper boundary condition on alpha - pc lower boundary condition type, - pc upper boundary condition type, - alpha lower boundary condition type, - alpha upper boundary condition type.

Definition at line 171 of file PSD.cpp.

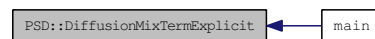
References ParamStructure::PSD::approximationMethod, VF::G(), PSD_parameters, and GridElement::size.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.6 void PSD::Initialize (ParamStructure::PSD parameters, Grid & grid, Matrix2D< double > L_UpperBoundaryCondition = Matrix2D<double> ())

Initializing: storing parameters, loading initial values, making boundary conditions, initializing output parameters. Simply, it is a creation of the object.

Parameters:

parameters - interpolation parameters structure

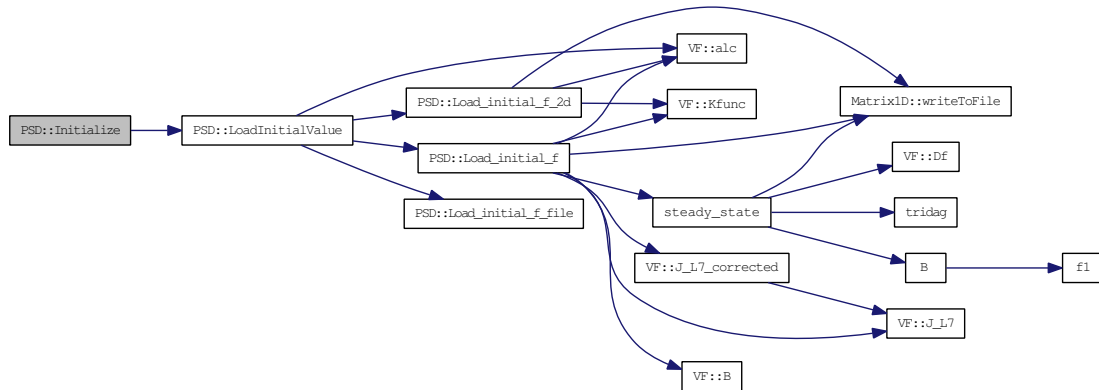
&grid - grid

Definition at line 82 of file PSD.cpp.

References LoadInitialValue(), ParamStructure::PSD::output_PSD_fileName4D, ParamStructure::PSD::output_PSD_folderName, output_without_grid_file, and PSD_parameters.

Referenced by PSD().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.7 void PSD::Interpolate (PSD & *otherPSD*, ParamStructure::Interpolation *interpolationParamStructure*, Grid & *oldGrid*, Grid & *newGrid*, Matrix2D< double > *newGrid_pc_lowerBoundaryCondition*, Matrix2D< double > *newGrid_pc_upperBoundaryCondition*)

Interpolation function.

It does interpolation... for log(function) or for just function, depends on parameters.

Parameters:

&otherPSD - PSD to interpolato the values from

interpolationParamStructure -

&oldGrid - old grid

&newGrid - new grid

newGrid_pc_lowerBoundaryCondition - upper energy boundary VALUE

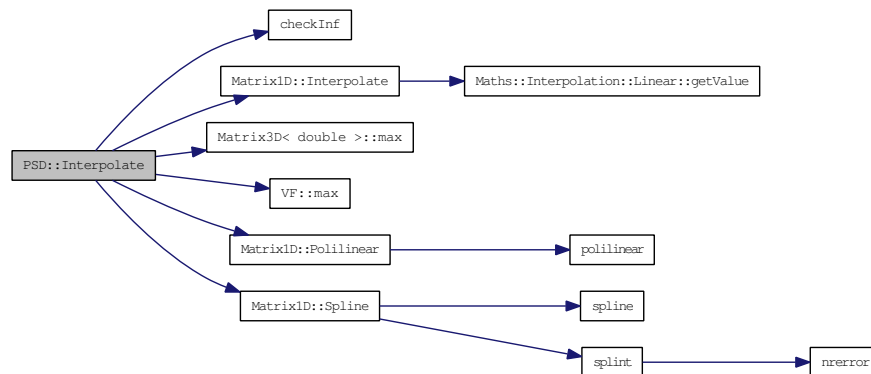
newGrid_pc_upperBoundaryCondition - lower energy boundary VALUE

Definition at line 1000 of file PSD.cpp.

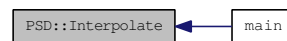
References Grid::alpha, checkInf(), Matrix1D< T >::Interpolate(), Grid::L, ParamStructure::Interpolation::linearSplineCoef, Matrix3D< double >::max(), VF::max(), ParamStructure::Interpolation::maxSecondDerivative, Grid::pc, Matrix1D< T >::Polilinear(), GridElement::size, Matrix1D< T >::Spline(), Grid::type, ParamStructure::Interpolation::type, and ParamStructure::Interpolation::useLog.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.8 void PSD::Load_initial_f (GridElement &L, GridElement &pc, GridElement &alpha, double tau, double Kp, Matrix2D< double > L_UpperBoundaryCondition, double min_f = 1.e-99, double fb_out = 1, double fb_in = 0)

Calculate initial PSD from steady state using boundary conditions.

Parameters:

&L - grid element L

&pc - grid element pc

&alpha - grid element alpha

tau - life time

Kp - Kp value

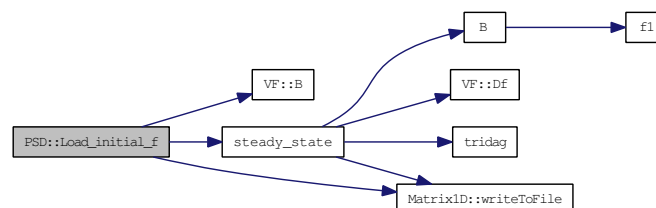
min_f - minimum of fuction (should be some positive value)

J_L7_function - parameter, J_L7 function. Can be J_L7 or J_L7_corrected

Definition at line 1360 of file PSD.cpp.

References VF::B(), i, GridElement::size, Matrix1D< T >::size_x, steady_state(), and Matrix1D< T >::writeToFile().

Here is the call graph for this function:



8.22.3.9 void PSD::Load_initial_f (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, double *tau*, double *Kp*, double *min_f* = 1.e-99, string *J_L7_function* = "J_L7", double *fb_out* = 1, double *fb_in* = 0)

Calculate initial PSD from steady state.

Parameters:

&*L* - grid element *L*

&*pc* - grid element *pc*

&*alpha* - grid element *alpha*

tau - life time

Kp - *Kp* value

min_f - minimum of function (should be some positive value)

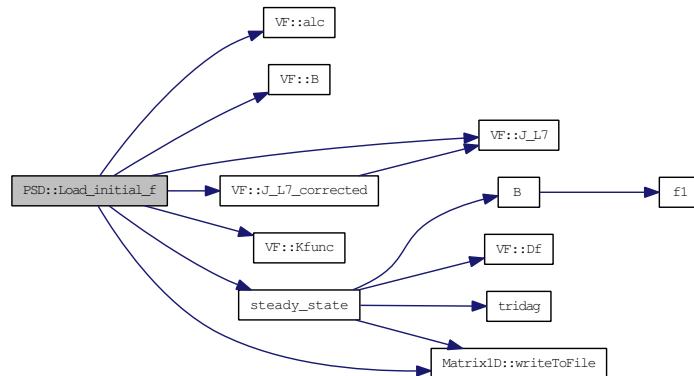
J_L7_function - parameter, *J_L7* function. Can be *J_L7* or *J_L7_corrected*

Definition at line 1287 of file PSD.cpp.

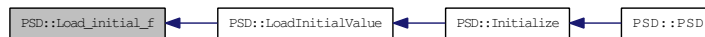
References VF::alc(), VF::B(), i, VF::J_L7(), VF::J_L7_corrected(), VF::Kfunc(), GridElement::size, Matrix1D< T >::size_x, steady_state(), and Matrix1D< T >::writeToFile().

Referenced by LoadInitialValue().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.10 void PSD::Load_initial_f_2d (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, const char * *filename*)

Load initial PSD from 2d-file (for 2d calculations).

Parameters:

&*L* - grid element *L*

&pc - grid element pc

&alpha - grid element alpha

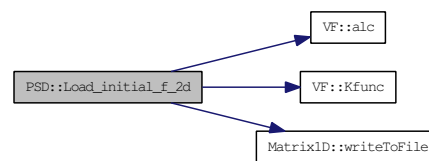
***filename** - file name

Definition at line 1402 of file PSD.cpp.

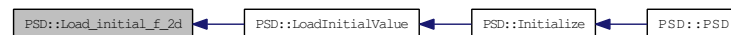
References VF::alc(), GridElement::GridElement_parameters, i, VF::Kfunc(), ParamStructure::GridElement::max, ParamStructure::GridElement::min, GridElement::size, and Matrix1D< T >::writeToFile().

Referenced by LoadInitialValue().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.11 void PSD::Load_initial_f_file (GridElement & L, GridElement & epc, GridElement & alpha, const char * filename, bool withGrid)

Load initial PSD from file.

Parameters:

&L - grid element L

&pc - grid element pc

&alpha - grid element alpha

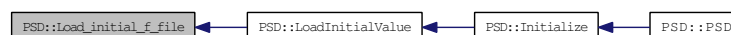
filename - filename

Definition at line 1243 of file PSD.cpp.

References err, and GridElement::size.

Referenced by LoadInitialValue().

Here is the caller graph for this function:



8.22.3.12 void PSD::LoadInitialValue (ParamStructure::PSD *parameters*, Grid & *grid*, Matrix2D< double > *L_UpperBoundaryCondition* = Matrix2D<double>())

Loading initial values - from a file or other sources.

This procedure calles other depends on initialPSDType.

Parameters:

parameters - initial PSD parameters structure

&grid - grid

moved to parameters.cpp if (parameters.initial_PSD_tauSteadyState <= 1.e-99) parameters.initial_PSD_tauSteadyState = 4.0/parameters.initial_PSD_Kp0;

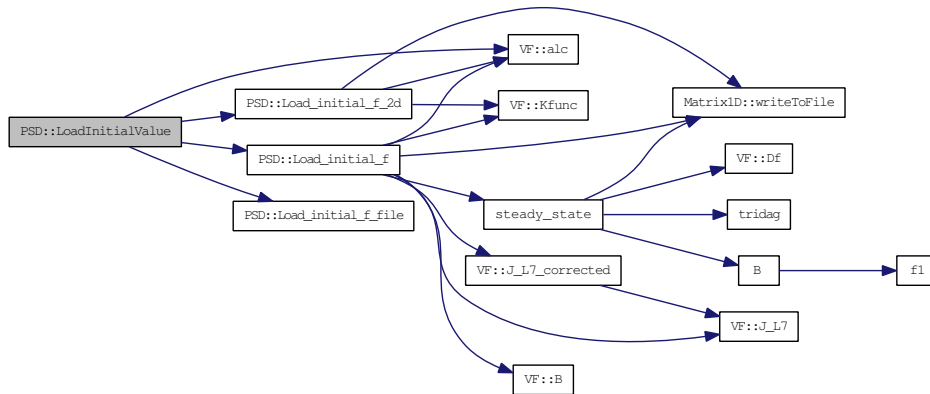
moved to parameters.cpp if (parameters.initial_PSD_tauSteadyState <= 1.e-99) parameters.initial_PSD_tauSteadyState = 4.0/parameters.initial_PSD_Kp0;

Definition at line 1149 of file PSD.cpp.

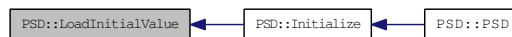
References VF::alc(), Grid::alpha, Grid::epc, ParamStructure::PSD::initial_PSD_fileName, ParamStructure::PSD::initial_PSD_inner_psd, ParamStructure::PSD::initial_PSD_J_L7_function, ParamStructure::PSD::initial_PSD_Kp0, ParamStructure::PSD::initial_PSD_outer_psd, ParamStructure::PSD::initial_PSD_some_constant_value, ParamStructure::PSD::initial_PSD_tauSteadyState, ParamStructure::PSD::initial_PSD_Type, Matrix2D< T >::initialized, Grid::L, Load_initial_f(), Load_initial_f_2d(), Load_initial_f_file(), Grid::pc, and GridElement::size.

Referenced by Initialize().

Here is the call graph for this function:



Here is the caller graph for this function:



8.22.3.13 void PSD::Output_without_grid (double *time*)

PSD output.

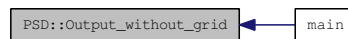
Parameters:

time - time

Definition at line 1222 of file PSD.cpp.

Referenced by main().

Here is the caller graph for this function:



8.22.3.14 void PSD::SourcesAndLosses (GridElement & *L*, GridElement & *pc*, GridElement & *alpha*, Matrix3D< double > & *SL*, double *dt*, double *Lpp*, double *tau*, double *tauLpp*)

Sources and losses term from the FP eq.

Parameters:

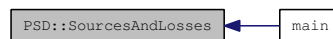
&*SL* - sources/losses 3D matrix

Definition at line 104 of file PSD.cpp.

References GridElement::size.

Referenced by main().

Here is the caller graph for this function:



8.22.4 Member Data Documentation

8.22.4.1 ofstream* PSD::output_without_grid_file

Definition at line 130 of file PSD.h.

Referenced by Initialize().

8.22.4.2 ParamStructure::PSD PSD::PSD_parameters

Definition at line 39 of file PSD.h.

Referenced by Diffusion_pc_alpha(), DiffusionMixTermExplicit(), and Initialize().

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

- [PSD.h](#)
- [PSD.cpp](#)

8.23 ParamStructure::PSD Struct Reference

PSD parameters structure.

```
#include <Parameters.h>
```

Public Attributes

- string [initial_PSD_Type](#)
Initial PSD parameters structure.
- string [initial_PSD_fileName](#)
File name, if we need to load initial values from file.
- double [initial_PSD_tauSteadyState](#)
Tau for steady state, if we are calculation initial values as steady state solytion.
- double [initial_PSD_Kp0](#)
Kp for steady state?
- double [initial_PSD_some_constant_value](#)
*Some psd value. Used as initial *PSD* value everywhere if *initial_PSD_Type* = *IPSDT_CONSTANT* or as minimal initial *PSD* value in other cases. Also used for some inner belt imitation if *initial_PSD_Type* = *IPSDT_STEADY_STATE*.*
- string [initial_PSD_J_L7_function](#)
Name of the function for flux at L=7. Can be 'J_L7' or 'J_L7_corrected'.
- double [initial_PSD_outer_psd](#)
*Outer *PSD* boundary value for steady state, default 1.*
- double [initial_PSD_inner_psd](#)
*Inner *PSD* boundary value for steady state, default 0.*
- string [output_PSD_folderName](#)
PSD output parameters structure
- string [output_PSD_fileName4D](#)
File name for psd output.
- double [output_PSD_timeStep](#)
Time step for psd output.
- string [approximationMethod](#)
*Approximation method. Check *StrToVal(string input, ApproximationMethods &place)* for known values.*
- string [solutionMethod](#)
*Solution method. Check *StrToVal(string input, SolutionMethods &place)* for known values.*
- int [SOL_maxiter](#)

- int [SOL_i_max](#)
Maximum number of steps, for iteration methods.
- double [SOL_max_iter_err](#)
For GMRES - number of iterations before restart.

8.23.1 Detailed Description

PSD parameters structure.

Definition at line 168 of file Parameters.h.

8.23.2 Member Data Documentation

8.23.2.1 string ParamStructure::PSD::approximationMethod

Approximation method. Check StrToVal(string input, ApproximationMethods &place) for known values.

Definition at line 187 of file Parameters.h.

Referenced by PSD::DiffusionMixTermExplicit(), ParamStructure::Load_parameters(), and main().

8.23.2.2 string ParamStructure::PSD::initial_PSD_fileName

File name, if we need to load initial values from file.

Definition at line 172 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.3 double ParamStructure::PSD::initial_PSD_inner_psd

Inner [PSD](#) boundary value for steady state, default 0.

Definition at line 178 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.4 string ParamStructure::PSD::initial_PSD_J_L7_function

Name of the function for flux at L=7. Can be 'J_L7' or 'J_L7_corrected'.

Definition at line 176 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.5 double ParamStructure::PSD::initial_PSD_Kp0

Kp for steady state?

Definition at line 174 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.6 double ParamStructure::PSD::initial_PSD_outer_psd

Outer [PSD](#) boundary value for steady state, default 1.

Definition at line 177 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.7 double ParamStructure::PSD::initial_PSD_some_constant_value

Some psd value. Used as initial [PSD](#) value everywhere if initial_PSD_Type = IPSDT_CONSTANT or as minimal initial [PSD](#) value in other cases. Also used for some inner belt imitation if initial_PSD_Type = IPSDT_STEADY_STATE.

Definition at line 175 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.8 double ParamStructure::PSD::initial_PSD_tauSteadyState

Tau for steady state, if we are calculation initial values as steady state solytion.

Definition at line 173 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), and PSD::LoadInitialValue().

8.23.2.9 string ParamStructure::PSD::initial_PSD_Type

Initial PSD parameters structure.

Tells us where to get initial PSD values. Check StrToVal(string input, InitialPSDTypes &place) for known values.

Definition at line 171 of file Parameters.h.

Referenced by ParamStructure::Load_parameters(), PSD::LoadInitialValue(), and PSD::PSD().

8.23.2.10 string ParamStructure::PSD::output_PSD_fileName4D

File name for psd output.

Definition at line 183 of file Parameters.h.

Referenced by PSD::Initialize(), and ParamStructure::Load_parameters().

8.23.2.11 string ParamStructure::PSD::output_PSD_folderName

PSD output parameters structure

Folder name for psd output.

Definition at line 182 of file Parameters.h.

Referenced by PSD::Initialize(), and ParamStructure::Load_parameters().

8.23.2.12 double ParamStructure::PSD::output_PSD_timeStep

Time step for psd output.

Definition at line 184 of file Parameters.h.

Referenced by ParamStructure::Load_parameters().

8.23.2.13 int ParamStructure::PSD::SOL_i_max

Maximum number of steps, for iteration methods.

Definition at line 192 of file Parameters.h.

Referenced by PSD::Diffusion_pc_alpha(), and ParamStructure::Load_parameters().

8.23.2.14 double ParamStructure::PSD::SOL_max_iter_err

For GMRES - number of iterations before restart.

Definition at line 193 of file Parameters.h.

Referenced by PSD::Diffusion_pc_alpha(), and ParamStructure::Load_parameters().

8.23.2.15 int ParamStructure::PSD::SOL_maxiter

Definition at line 191 of file Parameters.h.

Referenced by PSD::Diffusion_pc_alpha(), and ParamStructure::Load_parameters().

8.23.2.16 string ParamStructure::PSD::solutionMethod

Solution method. Check StrToVal(string input, SolutionMethods &place) for known values.

Definition at line 189 of file Parameters.h.

Referenced by PSD::Diffusion_pc_alpha(), and ParamStructure::Load_parameters().

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

- [Parameters.h](#)

8.24 `single_error` Class Reference

Hold some information about error in the code.

```
#include <error.h>
```

Public Member Functions

- `single_error` (string `code`)
Constructor.
- `single_error` (string `code`, string `msg`)
Constructor.

Public Attributes

- string `code`
Error code.
- string `msg`
Error message.

8.24.1 Detailed Description

Hold some information about error in the code.

Definition at line 26 of file `error.h`.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 `single_error::single_error` (string `code`) `[inline]`

Constructor.

Definition at line 35 of file `error.h`.

References `msg`.

8.24.2.2 `single_error::single_error` (string `code`, string `msg`) `[inline]`

Constructor.

Definition at line 42 of file `error.h`.

8.24.3 Member Data Documentation

8.24.3.1 string `single_error::code`

Error code.

Definition at line 29 of file `error.h`.

8.24.3.2 `string single_error::msg`

Error message.

Definition at line 32 of file `error.h`.

Referenced by `single_error()`.

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

- [error.h](#)

8.25 ParamStructure::SL Struct Reference

Sources and losses.

```
#include <Parameters.h>
```

Public Attributes

- bool [SL_L_top](#)
- string [SL_L_top_filename](#)
- bool [SL_E_min](#)
- string [SL_E_min_filename](#)

8.25.1 Detailed Description

Sources and losses.

Definition at line 201 of file Parameters.h.

8.25.2 Member Data Documentation

8.25.2.1 bool ParamStructure::SL::SL_E_min

Definition at line 204 of file Parameters.h.

Referenced by SourcesAndLosses::Initialize(), ParamStructure::Load_parameters(), and main().

8.25.2.2 string ParamStructure::SL::SL_E_min_filename

Definition at line 205 of file Parameters.h.

Referenced by SourcesAndLosses::Initialize(), and ParamStructure::Load_parameters().

8.25.2.3 bool ParamStructure::SL::SL_L_top

Definition at line 202 of file Parameters.h.

Referenced by SourcesAndLosses::Initialize(), ParamStructure::Load_parameters(), and main().

8.25.2.4 string ParamStructure::SL::SL_L_top_filename

Definition at line 203 of file Parameters.h.

Referenced by SourcesAndLosses::Initialize(), and ParamStructure::Load_parameters().

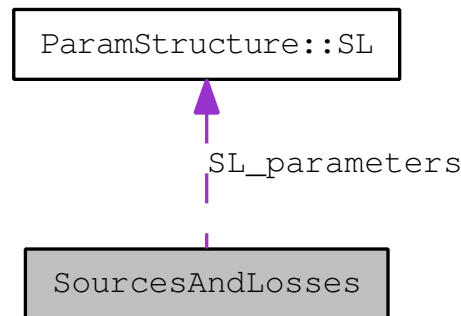
The documentation for this struct was generated from the following file:

- [Parameters.h](#)

8.26 SourcesAndLosses Class Reference

```
#include <SourcesAndLosses.h>
```

Collaboration diagram for SourcesAndLosses:



Public Member Functions

- [SourcesAndLosses](#) ()
- [SourcesAndLosses](#) ([ParamStructure::SL](#) parameters, [Grid](#) &grid, int numberOfIterations, double timeStep)
- void [Initialize](#) ([ParamStructure::SL](#) parameters, [Grid](#) &grid, int numberOfIterations, double timeStep)

Loading (or just calculate) sources and losses.

Public Attributes

- [ParamStructure::SL](#) SL_parameters

8.26.1 Detailed Description

[Todo](#)

Move loss cone losses to the rest of the sources and losses

Definition at line 19 of file SourcesAndLosses.h.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 [SourcesAndLosses::SourcesAndLosses](#) () `[inline]`

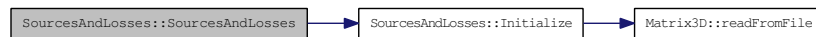
Definition at line 26 of file SourcesAndLosses.h.

8.26.2.2 [SourcesAndLosses::SourcesAndLosses](#) ([ParamStructure::SL](#) parameters, [Grid](#) & grid, int numberOfIterations, double timeStep)

Definition at line 13 of file SourcesAndLosses.cpp.

References Initialize().

Here is the call graph for this function:



8.26.3 Member Function Documentation

8.26.3.1 void SourcesAndLosses::Initialize (ParamStructure::SL *parameters*, Grid & *grid*, int *numberOfIterations*, double *timeStep*)

Loading (or just calculate) sources and losses.

Parameters:

parameters - PSD parameters

Todo

Add time-array, so loading functions can check time steps and grid.

Todo

Add time-array, so loading functions can check time steps and grid.

Definition at line 21 of file SourcesAndLosses.cpp.

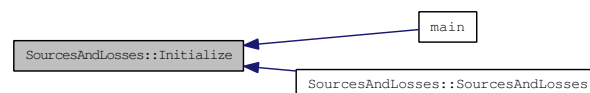
References Grid::alpha, Grid::epc, err, Grid::L, Grid::pc, Matrix3D< T >::readFromFile(), GridElement::size, ParamStructure::SL::SL_E_min, ParamStructure::SL::SL_E_min_filename, ParamStructure::SL::SL_L_top, ParamStructure::SL::SL_L_top_filename, and SL_parameters.

Referenced by main(), and SourcesAndLosses().

Here is the call graph for this function:



Here is the caller graph for this function:



8.26.4 Member Data Documentation

8.26.4.1 ParamStructure::SL SourcesAndLosses::SL_parameters

Definition at line 22 of file SourcesAndLosses.h.

Referenced by Initialize().

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

- [SourcesAndLosses.h](#)
- [SourcesAndLosses.cpp](#)

Chapter 9

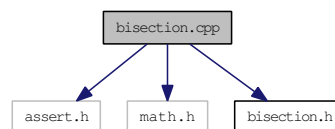
File Documentation

9.1 bisection.cpp File Reference

Bisection method of root finding code.

```
#include <assert.h>
#include <math.h>
#include "bisection.h"
```

Include dependency graph for bisection.cpp:



Functions

- double [bisection](#) (double(*f)(double, double), double Alpha, double x0, double x1, double eps)

9.1.1 Detailed Description

Bisection method of root finding code.

Author:

Unknown (Numeric recepies?)

Definition in file [bisection.cpp](#).

9.1.2 Function Documentation

9.1.2.1 `double bisection (double(*) (double, double) f, double Alpha, double x0, double x1, double eps)`

Looking for roots of function f for given Alpha on the interval x0-x1

Parameters:

double (*f)(double x, double Alpha), - Function for root finding. Only x-roots are found, Alpha is constant

double Alpha, - constant for the f function

double x0 = -1E+7, - left boundary of the interval

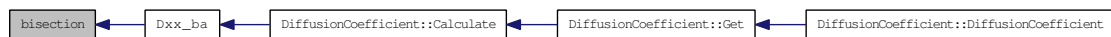
double x1 = 1E+7, - right boundary of the interval

double eps = max_error; - max error

Definition at line 22 of file bisection.cpp.

Referenced by Dxx_ba().

Here is the caller graph for this function:

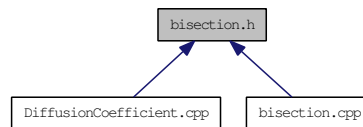


9.2 bisection.d File Reference

9.3 bisection.h File Reference

Bisection method of root finding header.

This graph shows which files directly or indirectly include this file:



Defines

- #define `max_error` 1E-15
max_error maximum error after calculation

Functions

- double `bisection` (double(*f)(double x, double Alpha), double Alpha, double x0=-1E+7, double x1=1E+7, double eps=max_error)

9.3.1 Detailed Description

Bisection method of root finding header.

Author:

Unknown (Numeric recepies?)

Definition in file `bisection.h`.

9.3.2 Define Documentation

9.3.2.1 #define max_error 1E-15

`max_error` maximum error after calculation

Definition at line 13 of file `bisection.h`.

9.3.3 Function Documentation

9.3.3.1 double bisection (double(*)(double x, double Alpha)f, double Alpha, double x0 = -1E+7, double x1 = 1E+7, double eps = max_error)

Looking for roots of function f for given Alpha on the interval x0-x1

Parameters:

double (*f)(double x, double Alpha), - Function for root finding. Only x-roots are found, Alpha is constant

double Alpha, - constant for the f function

double x0 = -1E+7, - left boundary of the interval

double x1 = 1E+7, - right boundary of the interval

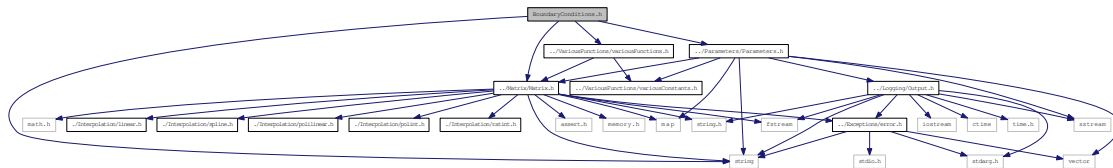
double eps = max_error); - max error

9.5 BoundaryConditions.d File Reference

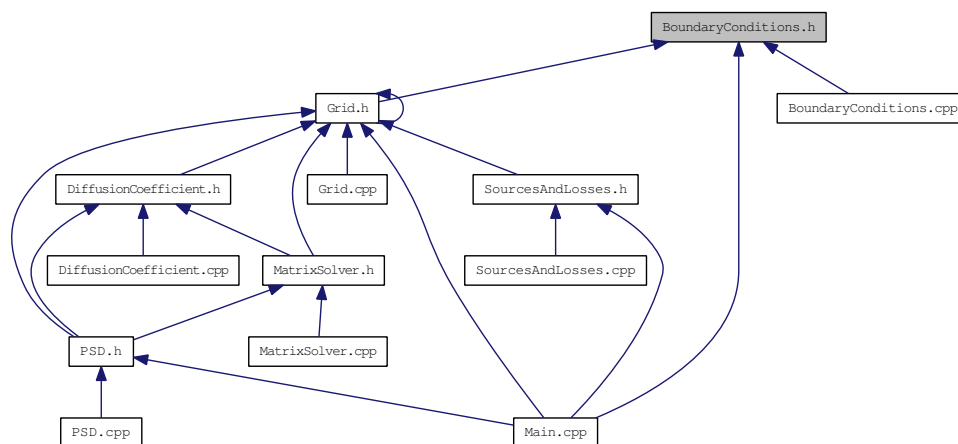
9.6 BoundaryConditions.h File Reference

```
#include <string>
#include "../Matrix/Matrix.h"
#include "../Parameters/Parameters.h"
#include "../VariousFunctions/variousFunctions.h"
```

Include dependency graph for BoundaryConditions.h:



This graph shows which files directly or indirectly include this file:



Classes

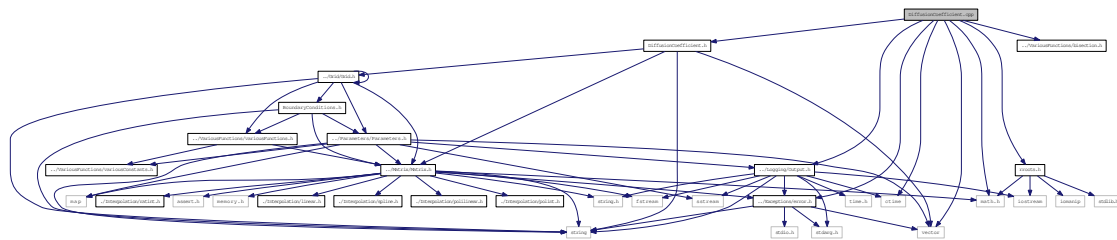
- class [BoundaryCondition](#)

9.7 DiffusionCoefficient.cpp File Reference

Diffusion coefficients calculation, loading, activating, scaling etc code.

```
#include "DiffusionCoefficient.h"
#include <math.h>
#include <vector>
#include "rrroots.h"
#include "../VariousFunctions/bisection.h"
#include "../Logging/Output.h"
#include "../Exceptions/error.h"
#include <ctime>
```

Include dependency graph for DiffusionCoefficient.cpp:



Defines

- `#define double_zero 1.e-21`
- `#define min_Dxx 1.e-21`

Functions

- double [Alpha_ne](#) (double pangle, double lambda, double L)
- double [f1](#) (double lambda)
- double [B](#) (double lambda, double L)
- double [func_tmp](#) (double x, double Alpha)
- double [F_cap](#) (double x, double y, double b, double s, double epsilon, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [F_cap2](#) (double x, double y, double a, double beta, double mu, double s, double epsilon, double Alpha_star, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [quad1](#) (double(*func)(double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure), double a, double b, int M, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Dxx_ba](#) (double L1, double EMeV, double Alpha, double int_Dxx_loc(double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure), [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [int_Daa_loc](#) (double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [int_Dpp_loc](#) (double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [int_Dpa_loc](#) (double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure)

- double [Dxx_local](#) (double lambda, double Dxx_root(double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double Alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure), [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Daa_root](#) (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double Alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Dpa_root](#) (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double Alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Dpp_root](#) (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double Alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- std::vector< double > [rrouts](#) (double x_1, double x_2, double eta1, double eta2, double eta3, double epsilon, double beta, double mu, double Alpha_star, double a, [DiffusionCoefficientParamStructure](#) DxxParamStructure)

routs finding routine

9.7.1 Detailed Description

Diffusion coefficients calculation, loading, activating, scaling etc code.

Author:

Developed by Yuri Shprits

Definition in file [DiffusionCoefficient.cpp](#).

9.7.2 Define Documentation

9.7.2.1 #define double_zero 1.e-21

Definition at line 23 of file DiffusionCoefficient.cpp.

Referenced by rrouts().

9.7.2.2 #define min_Dxx 1.e-21

Definition at line 24 of file DiffusionCoefficient.cpp.

Referenced by DiffusionCoefficient::Calculate().

9.7.3 Function Documentation

9.7.3.1 double Alpha_ne (double *pangle*, double *lambda*, double *L*)

Definition at line 609 of file DiffusionCoefficient.cpp.

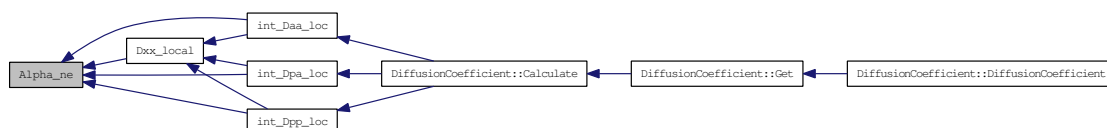
References f1().

Referenced by Dxx_local(), int_Daa_loc(), int_Dpa_loc(), and int_Dpp_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



9.7.3.2 double B (double *lambda*, double *L*)

Definition at line 617 of file DiffusionCoefficient.cpp.

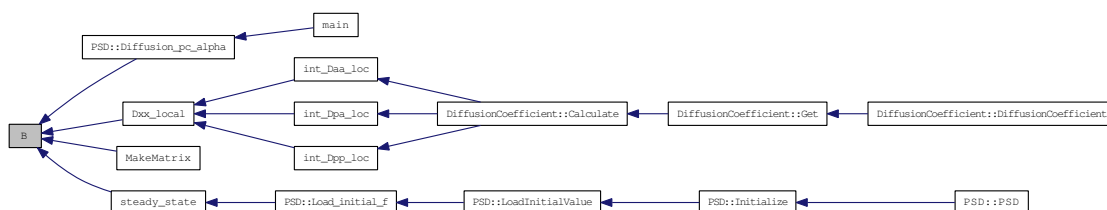
References f1().

Referenced by PSD::Diffusion_pc_alpha(), Dxx_local(), MakeMatrix(), and steady_state().

Here is the call graph for this function:



Here is the caller graph for this function:



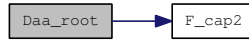
9.7.3.3 double Daa_root (double *Omega_e*, double *x*, double *mu*, double *su*, double *y*, double *beta*, double *a*, double *b*, double *Alpha_star*, double *s*, double *epsilon*, double *d_x*, double *x_m*, double *R*, DiffusionCoefficientParamStructure DxxParamStructure)

Definition at line 820 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::EMeV, F_cap2(), and DiffusionCoefficientParamStructure::nu.

Referenced by int_Daa_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



9.7.3.4 **double Dpa_root (double *Omega_e*, double *x*, double *mu*, double *su*, double *y*, double *beta*, double *a*, double *b*, double *Alpha_star*, double *s*, double *epsilon*, double *d_x*, double *x_m*, double *R*, DiffusionCoefficientParamStructure *DxxParamStructure*)**

Definition at line 827 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::EMeV, F_cap2(), and DiffusionCoefficientParamStructure::nu.

Referenced by int_Dpa_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



9.7.3.5 **double Dpp_root (double *Omega_e*, double *x*, double *mu*, double *su*, double *y*, double *beta*, double *a*, double *b*, double *Alpha_star*, double *s*, double *epsilon*, double *d_x*, double *x_m*, double *R*, DiffusionCoefficientParamStructure *DxxParamStructure*)**

Definition at line 834 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::EMeV, F_cap2(), and DiffusionCoefficientParamStructure::nu.

Referenced by int_Dpp_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



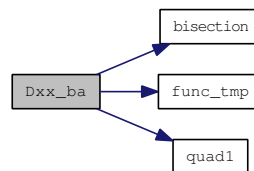
9.7.3.6 **double Dxx_ba** (double *L1*, double *EMeV*, double *Alpha*, double *int_Dxx_loc* double *lambda*, DiffusionCoefficientParamStructure *DxxParamStructure*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 657 of file DiffusionCoefficient.cpp.

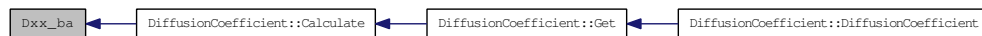
References DiffusionCoefficientParamStructure::Alpha, bisection(), DiffusionCoefficientParamStructure::EMeV, func_tmp(), DiffusionCoefficientParamStructure::L, DiffusionCoefficientParamStructure::lam_max, DiffusionCoefficientParamStructure::lam_min, DiffusionCoefficientParamStructure::nint, and quad1().

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



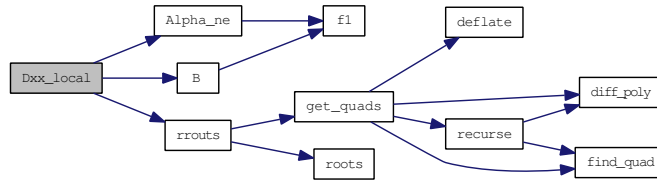
9.7.3.7 **double Dxx_local** (double *lambda*, double *Dxx_root* double *Omega_e*, double *x*, double *mu*, double *su*, double *y*, double *beta*, double *a*, double *b*, double *Alpha_star*, double *s*, double *epsilon*, double *d_x*, double *x_m*, double *R*, DiffusionCoefficientParamStructure *DxxParamStructure*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 694 of file DiffusionCoefficient.cpp.

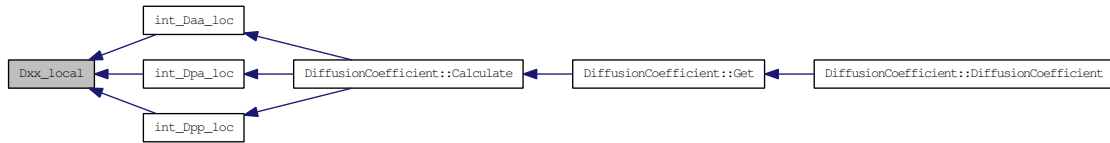
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), B(), DiffusionCoefficientParamStructure::Bw, DiffusionCoefficientParamStructure::BwFromLambda, DiffusionCoefficientParamStructure::d_omega, DiffusionCoefficientParamStructure::EMeV, DiffusionCoefficientParamStructure::eta1, DiffusionCoefficientParamStructure::eta2, DiffusionCoefficientParamStructure::eta3, DiffusionCoefficientParamStructure::f_i, DiffusionCoefficientParamStructure::L, DiffusionCoefficientParamStructure::numberDensity, DiffusionCoefficientParamStructure::omega_lc, DiffusionCoefficientParamStructure::Omega_m, DiffusionCoefficientParamStructure::Omega_mType, DiffusionCoefficientParamStructure::omega_uc, DiffusionCoefficientParamStructure::particle, rroots(), and DiffusionCoefficientParamStructure::s.

Referenced by `int_Daa_loc()`, `int_Dpa_loc()`, and `int_Dpp_loc()`.

Here is the call graph for this function:



Here is the caller graph for this function:

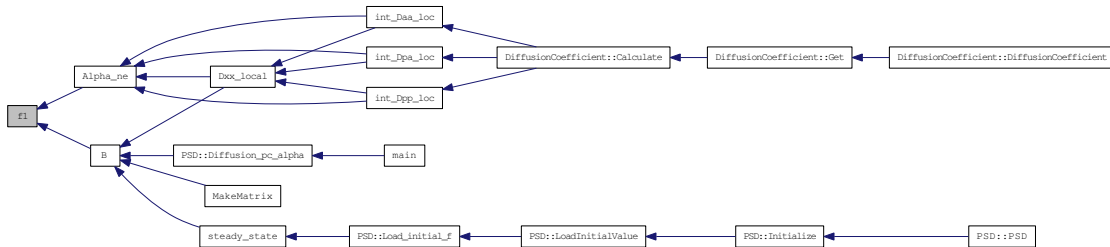


9.7.3.8 double f1 (double *lambda*)

Definition at line 613 of file DiffusionCoefficient.cpp.

Referenced by Alpha_ne(), and B().

Here is the caller graph for this function:



9.7.3.9 double F_cap (double *x*, double *y*, double *b*, double *s*, double *epsilon*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 625 of file DiffusionCoefficient.cpp.

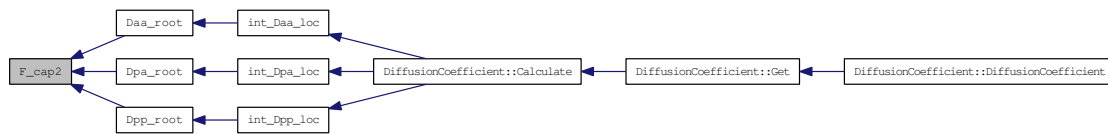
9.7.3.10 double F_cap2 (double *x*, double *y*, double *a*, double *beta*, double *mu*, double *s*, double *epsilon*, double *Alpha_star*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 634 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::eta1, DiffusionCoefficientParamStructure::eta2, and DiffusionCoefficientParamStructure::eta3.

Referenced by Daa_root(), Dpa_root(), and Dpp_root().

Here is the caller graph for this function:

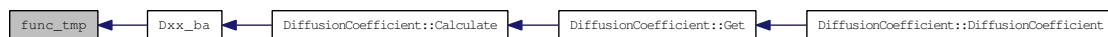


9.7.3.11 double func_tmp (double *x*, double *Alpha*)

Definition at line 621 of file DiffusionCoefficient.cpp.

Referenced by Dxx_ba().

Here is the caller graph for this function:



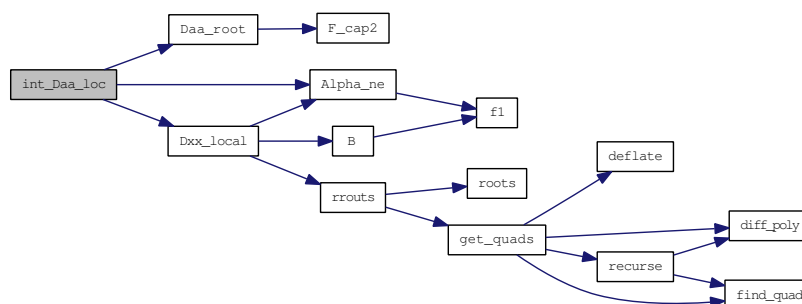
9.7.3.12 double int_Daa_loc (double *lambda*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 673 of file DiffusionCoefficient.cpp.

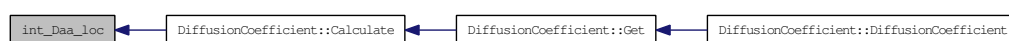
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), Daa_root(), Dxx_local(), and DiffusionCoefficientParamStructure::L.

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



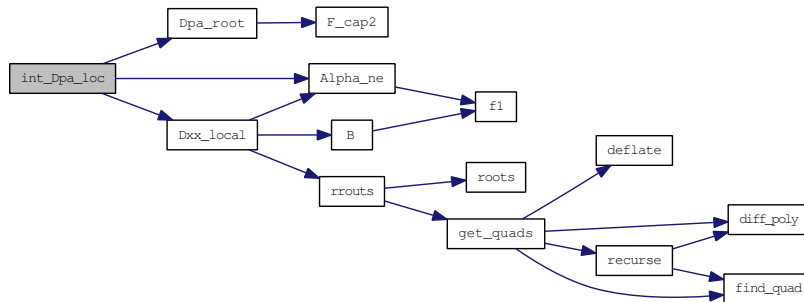
9.7.3.13 `double int_Dpa_loc (double lambda, DiffusionCoefficientParamStructure DxxParamStructure)`

Definition at line 687 of file DiffusionCoefficient.cpp.

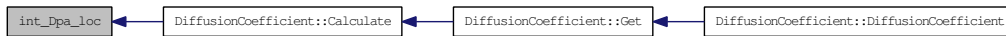
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), Dpa_root(), Dxx_local(), and DiffusionCoefficientParamStructure::L.

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



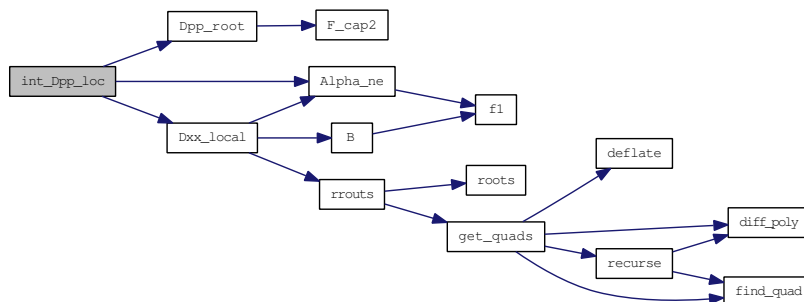
9.7.3.14 `double int_Dpp_loc (double lambda, DiffusionCoefficientParamStructure DxxParamStructure)`

Definition at line 680 of file DiffusionCoefficient.cpp.

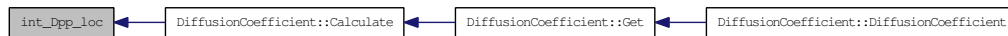
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), Dpp_root(), Dxx_local(), and DiffusionCoefficientParamStructure::L.

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:

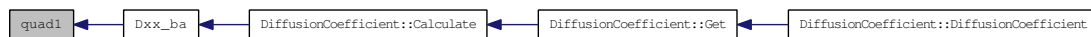


9.7.3.15 double quad1 (double(*) (double lambda, DiffusionCoefficientParamStructure DxxParamStructure) func, double a, double b, int M, DiffusionCoefficientParamStructure DxxParamStructure)

Definition at line 641 of file DiffusionCoefficient.cpp.

Referenced by Dxx_ba().

Here is the caller graph for this function:



9.7.3.16 std::vector<double> rrouts (double x_1, double x_2, double eta1, double eta2, double eta3, double epsilon, double beta, double mu, double Alpha_star, double a, DiffusionCoefficientParamStructure DxxParamStructure)

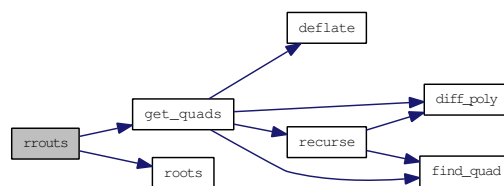
roots finding routine

Definition at line 845 of file DiffusionCoefficient.cpp.

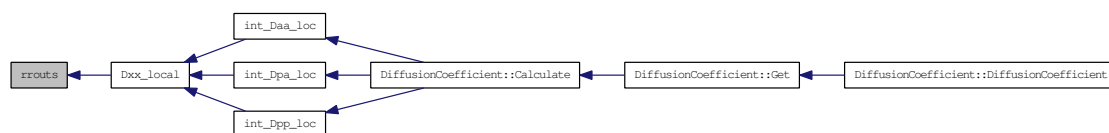
References double_zero, get_quads(), i, roots(), and DiffusionCoefficientParamStructure::s.

Referenced by Dxx_local().

Here is the call graph for this function:



Here is the caller graph for this function:



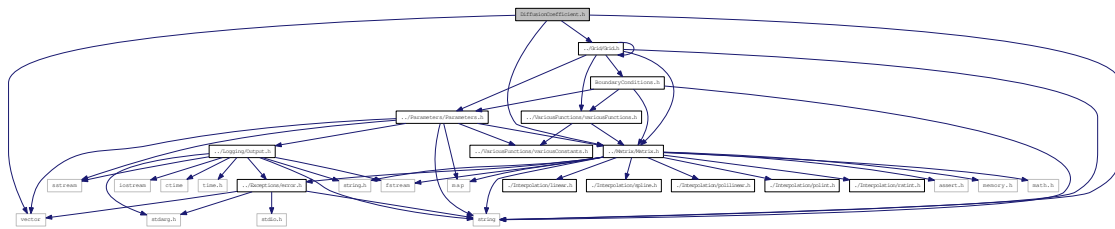
9.8 DiffusionCoefficient.d File Reference

9.9 DiffusionCoefficient.h File Reference

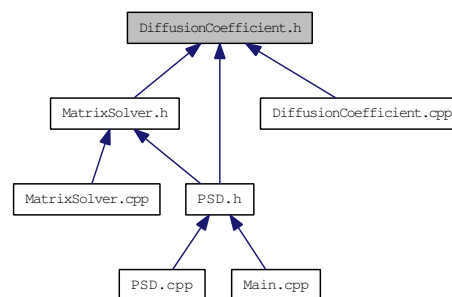
Diffusion coefficients calculation, loading, activating, scaling etc.

```
#include <string>
#include <vector>
#include "../Matrix/Matrix.h"
#include "../Grid/Grid.h"
```

Include dependency graph for DiffusionCoefficient.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [DiffusionCoefficient](#)

Class [DiffusionCoefficient](#) holds diffusion coefficient matrix and routines to load and calculate it.

- class [DiffusionCoefficientsGroup](#)

It has [DiffusionCoefficient](#) class as a parent class, so it stores there summation of all the coefficients to use in diffusion.

Functions

- double [Dxx_ba](#) (double L, double epc, double alpha, double int_Dxx_loc(double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure), [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Dxx_local](#) (double lambda, double Dxx_root(double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double alpha_star, double s, double epsilon, double

- d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure), [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Daa_root](#) (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Dpa_root](#) (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [Dpp_root](#) (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double alpha_star, double s, double epsilon, double d_x, double x_m, double R, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [int_Daa_loc](#) (double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [int_Dpp_loc](#) (double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [int_Dpa_loc](#) (double lambda, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- double [f1](#) (double lambda)
- double [Alpha_ne](#) (double pangle, double lambda, double L)
- double [func_tmp](#) (double x, double Alpha)
- double [F_cap](#) (double x, double y, double b, double s, double epsilon, [DiffusionCoefficientParamStructure](#) DxxParamStructure)
- std::vector< double > [rrouts](#) (double x_1, double x_2, double yida1, double yida2, double yida3, double epsilon, double beta, double mu, double alpha_star, double a, [DiffusionCoefficientParamStructure](#) DxxParamStructure)

rrouts finding routine

9.9.1 Detailed Description

Diffusion coefficients calculation, loading, activating, scaling etc.

Header file.

Author:

Developed by Yuri Shprits

Definition in file [DiffusionCoefficient.h](#).

9.9.2 Function Documentation

9.9.2.1 double Alpha_ne (double pangle, double lambda, double L)

Definition at line 609 of file DiffusionCoefficient.cpp.

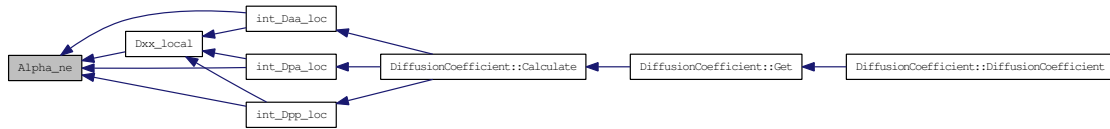
References [f1\(\)](#).

Referenced by [Dxx_local\(\)](#), [int_Daa_loc\(\)](#), [int_Dpa_loc\(\)](#), and [int_Dpp_loc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.2.2 double Daa_root (double *Omega_e*, double *x*, double *mu*, double *su*, double *y*, double *beta*, double *a*, double *b*, double *alpha_star*, double *s*, double *epsilon*, double *d_x*, double *x_m*, double *R*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 820 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::EMeV, F_cap2(), and DiffusionCoefficientParamStructure::nu.

Referenced by int_Daa_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.2.3 double Dpa_root (double *Omega_e*, double *x*, double *mu*, double *su*, double *y*, double *beta*, double *a*, double *b*, double *alpha_star*, double *s*, double *epsilon*, double *d_x*, double *x_m*, double *R*, DiffusionCoefficientParamStructure *DxxParamStructure*)

Definition at line 827 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::EMeV, F_cap2(), and DiffusionCoefficientParamStructure::nu.

Referenced by int_Dpa_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.2.4 `double Dpp_root (double Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double alpha_star, double s, double epsilon, double d_x, double x_m, double R, DiffusionCoefficientParamStructure DxxParamStructure)`

Definition at line 834 of file DiffusionCoefficient.cpp.

References DiffusionCoefficientParamStructure::EMeV, F_cap2(), and DiffusionCoefficientParamStructure::nu.

Referenced by int_Dpp_loc().

Here is the call graph for this function:



Here is the caller graph for this function:



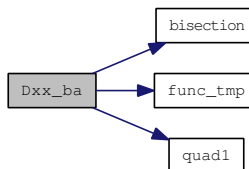
9.9.2.5 `double Dxx_ba (double L, double epc, double alpha, double int_Dxx_loc, double lambda, DiffusionCoefficientParamStructure DxxParamStructure, DiffusionCoefficientParamStructure DxxParamStructure)`

Definition at line 657 of file DiffusionCoefficient.cpp.

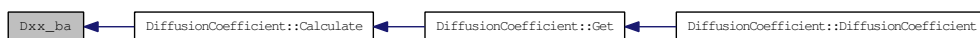
References DiffusionCoefficientParamStructure::Alpha, bisection(), DiffusionCoefficientParamStructure::EMeV, func_tmp(), DiffusionCoefficientParamStructure::L, DiffusionCoefficientParamStructure::lam_max, DiffusionCoefficientParamStructure::lam_min, DiffusionCoefficientParamStructure::nint, and quad1().

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



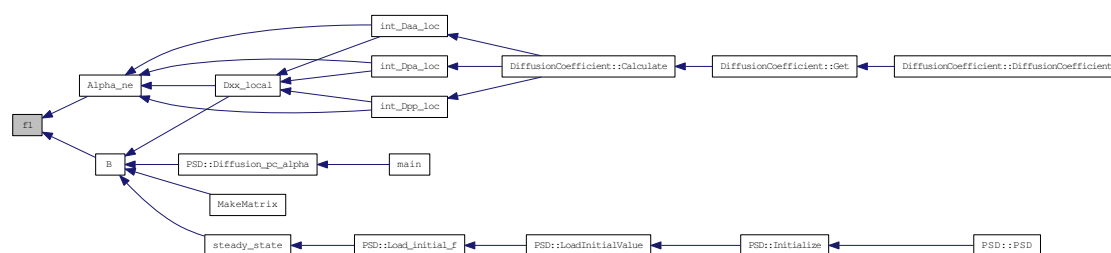
9.9.2.6 `double Dxx_local (double lambda, double Dxx_rootdouble Omega_e, double x, double mu, double su, double y, double beta, double a, double b, double alpha_star, double s, double epsilon, double d_x, double x_m, double R, DiffusionCoefficientParamStructure DxxParamStructure, DiffusionCoefficientParamStructure DxxParamStructure)`

9.9.2.7 `double f1 (double lambda)`

Definition at line 613 of file DiffusionCoefficient.cpp.

Referenced by Alpha_ne(), and B().

Here is the caller graph for this function:



9.9.2.8 `double F_cap (double x, double y, double b, double s, double epsilon, DiffusionCoefficientParamStructure DxxParamStructure)`

Definition at line 625 of file DiffusionCoefficient.cpp.

9.9.2.9 `double func_tmp (double x, double Alpha)`

Definition at line 621 of file DiffusionCoefficient.cpp.

Referenced by Dxx_ba().

Here is the caller graph for this function:



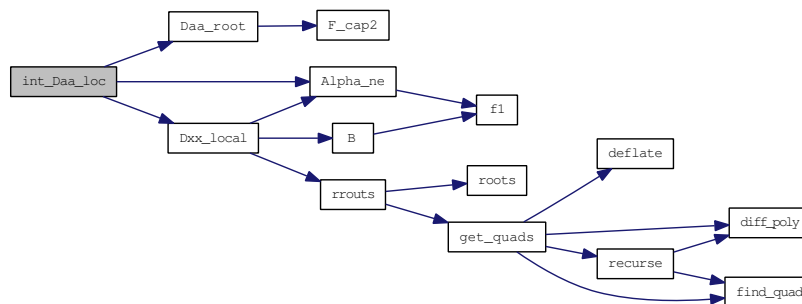
9.9.2.10 `double int_Daa_loc (double lambda, DiffusionCoefficientParamStructure DxxParamStructure)`

Definition at line 673 of file DiffusionCoefficient.cpp.

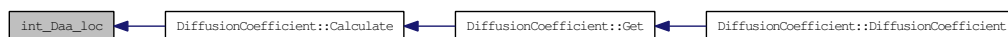
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), Daa_root(), Dxx_local(), and DiffusionCoefficientParamStructure::L.

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



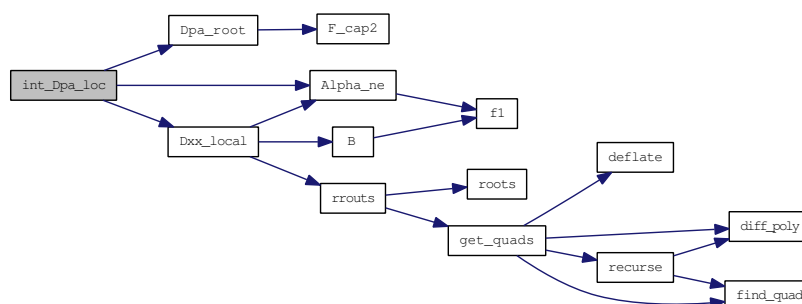
9.9.2.11 double int_Dpa_loc (double *lambda*, DiffusionCoefficientParamStructure DxxParamStructure)

Definition at line 687 of file DiffusionCoefficient.cpp.

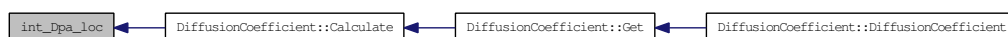
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), Dpa_root(), Dxx_local(), and DiffusionCoefficientParamStructure::L.

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



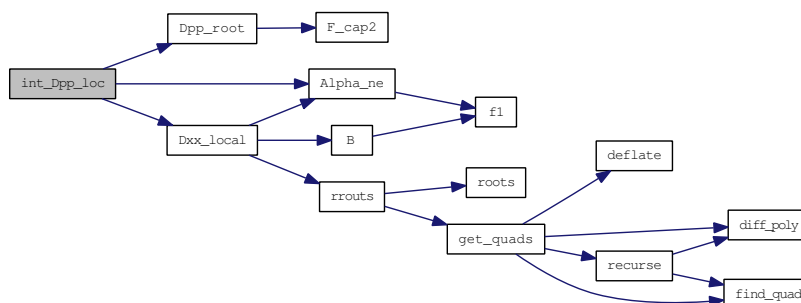
9.9.2.12 double int_Dpp_loc (double *lambda*, DiffusionCoefficientParamStructure DxxParamStructure)

Definition at line 680 of file DiffusionCoefficient.cpp.

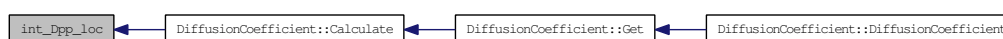
References DiffusionCoefficientParamStructure::Alpha, Alpha_ne(), Dpp_root(), Dxx_local(), and DiffusionCoefficientParamStructure::L.

Referenced by DiffusionCoefficient::Calculate().

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.2.13 std::vector<double> rrouts (double x_1, double x_2, double yida1, double yida2, double yida3, double epsilon, double beta, double mu, double alpha_star, double a, DiffusionCoefficientParamStructure DxxParamStructure)

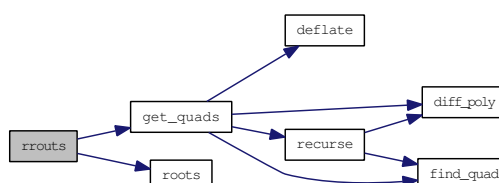
roots finding routine

Definition at line 845 of file DiffusionCoefficient.cpp.

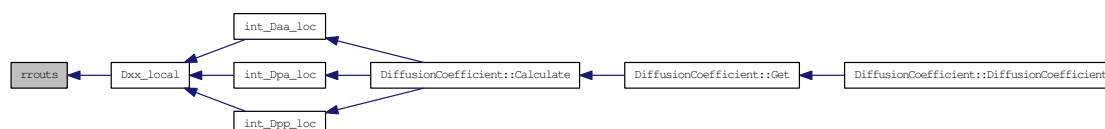
References double_zero, get_quads(), i, roots(), and DiffusionCoefficientParamStructure::s.

Referenced by Dxx_local().

Here is the call graph for this function:



Here is the caller graph for this function:

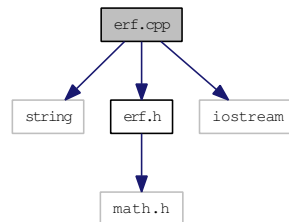


9.10 erf.cpp File Reference

Error function.

```
#include <string>
#include "erf.h"
#include <iostream>
```

Include dependency graph for erf.cpp:



Functions

- bool [str2bool](#) (string str)
Converting string to boolean.
- string [bool2str](#) (bool b)
Converting boolean to string.
- void [nerror](#) (const char *msg)
- double [gammln](#) (double xx)
Returns the value $\ln[\Gamma(xx)]$ for $xx > 0$.
- double [gamp](#) (double a, double x)
Returns the incomplete gamma function $P(a, x)$.
- double [gammq](#) (double a, double x)
Returns the incomplete gamma function $Q(a, x) = 1 - P(a, x)$.
- void [gser](#) (double *gamser, double a, double x, double *gln)
Returns the incomplete gamma function $P(a, x)$ evaluated by its series representation as gamser. Also returns $\ln \Gamma(a)$ as gln.
- void [gcf](#) (double *gammcf, double a, double x, double *gln)
Returns the incomplete gamma function $Q(a, x)$ evaluated by its continued fraction representation as gammcf. Also returns $\ln \Gamma(a)$ as gln.
- double [erf](#) (double x)
Returns the error function $\text{erf}(x)$.

9.10.1 Detailed Description

Error function.

Author:

Numerical Recepies.

Definition in file [erf.cpp](#).

9.10.2 Function Documentation

9.10.2.1 string bool2str (bool *b*)

Converting boolean to string.

Definition at line 522 of file Parameters.cpp.

9.10.2.2 double erf (double *x*)

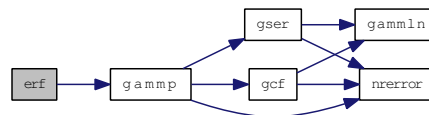
Returns the error function $\text{erf}(x)$.

Definition at line 137 of file erf.cpp.

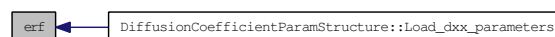
References `gammp()`.

Referenced by `DiffusionCoefficientParamStructure::Load_dxx_parameters()`.

Here is the call graph for this function:



Here is the caller graph for this function:



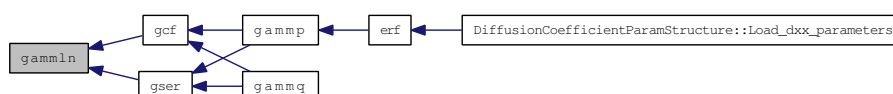
9.10.2.3 double gammln (double *xx*)

Returns the value $\ln[\Gamma(xx)]$ for $xx > 0$.

Definition at line 23 of file erf.cpp.

Referenced by `gcf()`, and `gser()`.

Here is the caller graph for this function:



9.10.2.4 double gammp (double *a*, double *x*)

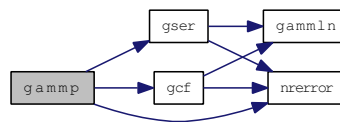
Returns the incomplete gamma function $P(a, x)$.

Definition at line 39 of file erf.cpp.

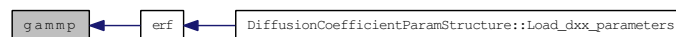
References gcf(), gser(), and nrerror().

Referenced by erf().

Here is the call graph for this function:



Here is the caller graph for this function:



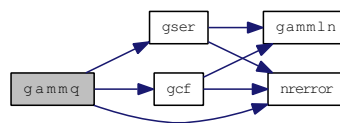
9.10.2.5 double gammq (double *a*, double *x*)

Returns the incomplete gamma function $Q(a, x) = 1 - P(a, x)$.

Definition at line 56 of file erf.cpp.

References gcf(), gser(), and nrerror().

Here is the call graph for this function:



9.10.2.6 void gcf (double * *gammcf*, double *a*, double *x*, double * *gln*)

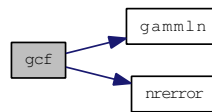
Returns the incomplete gamma function $Q(a, x)$ evaluated by its continued fraction representation as `gammcf`. Also returns $\ln(a)$ as `gln`.

Definition at line 103 of file erf.cpp.

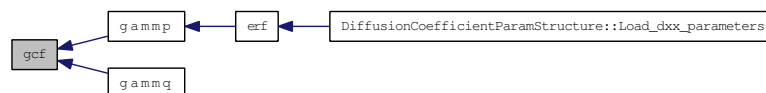
References EPS, FPMIN, `gammln()`, `i`, `ITMAX`, and `nrerror()`.

Referenced by `gamm()`, and `gammq()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.7 void gser (double * *gamser*, double *a*, double *x*, double * *gln*)

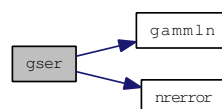
Returns the incomplete gamma function $P(a, x)$ evaluated by its series representation as *gamser*. Also returns $\ln \Gamma(a)$ as *gln*.

Definition at line 74 of file erf.cpp.

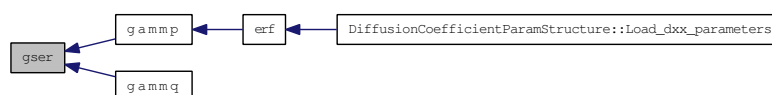
References EPS, `gammln()`, ITMAX, and `nrerror()`.

Referenced by `gammp()`, and `gammq()`.

Here is the call graph for this function:



Here is the caller graph for this function:

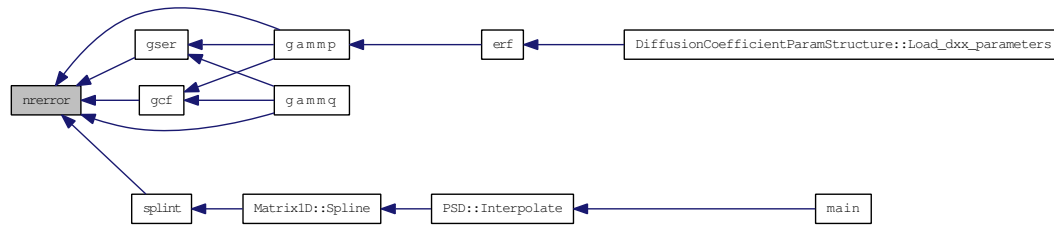


9.10.2.8 void nrerror (const char * *msg*)

Definition at line 17 of file erf.cpp.

Referenced by `gammp()`, `gammq()`, `gcf()`, `gser()`, and `splint()`.

Here is the caller graph for this function:



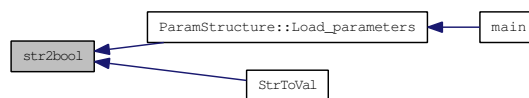
9.10.2.9 bool str2bool (string str)

Converting string to boolean.

Definition at line 514 of file Parameters.cpp.

Referenced by ParamStructure::Load_parameters(), and StrToVal().

Here is the caller graph for this function:



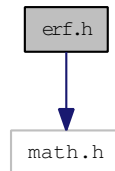
9.11 erf.d File Reference

9.12 erf.h File Reference

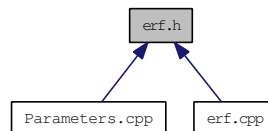
Error function.

```
#include <math.h>
```

Include dependency graph for erf.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define [ITMAX](#) 100000
Maximum allowed number of iterations.
- #define [EPS](#) 3.0e-7
Relative accuracy.
- #define [FPMIN](#) 1.0e-30
Number near the smallest representable doubleing-point number.

Functions

- double [gammln](#) (double xx)
Returns the value $\ln[?(xx)]$ for $xx > 0$.
- double [gammp](#) (double a, double x)
Returns the incomplete gamma function $P(a, x)$.
- double [gammq](#) (double a, double x)
Returns the incomplete gamma function $Q(a, x) ? 1 ? P(a, x)$.
- void [gser](#) (double *gamser, double a, double x, double *gln)
Returns the incomplete gamma function $P(a, x)$ evaluated by its series representation as gamser. Also returns $\ln ?(a)$ as gln.

- void [gcf](#) (double *gammcf, double a, double x, double *gln)
Returns the incomplete gamma function $Q(a, x)$ evaluated by its continued fraction representation as gammcf. Also returns $\ln(a)$ as gln.
- double [erf](#) (double x)
Returns the error function $\text{erf}(x)$.

9.12.1 Detailed Description

Error function.

Author:

Numerical recepies.

Definition in file [erf.h](#).

9.12.2 Define Documentation

9.12.2.1 #define EPS 3.0e-7

Relative accuracy.

Definition at line 13 of file erf.h.

Referenced by [gcf\(\)](#), and [gser\(\)](#).

9.12.2.2 #define FPMIN 1.0e-30

Number near the smallest representable doubleing-point number.

Definition at line 15 of file erf.h.

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

9.12.2.3 #define ITMAX 100000

Maximum allowed number of iterations.

Definition at line 11 of file erf.h.

Referenced by [gcf\(\)](#), and [gser\(\)](#).

9.12.3 Function Documentation

9.12.3.1 double erf (double x)

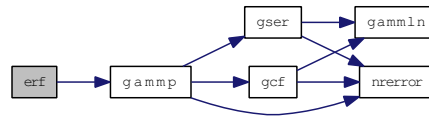
Returns the error function $\text{erf}(x)$.

Definition at line 137 of file erf.cpp.

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

Referenced by `DiffusionCoefficientParamStructure::Load_dxx_parameters()`.

Here is the call graph for this function:



Here is the caller graph for this function:



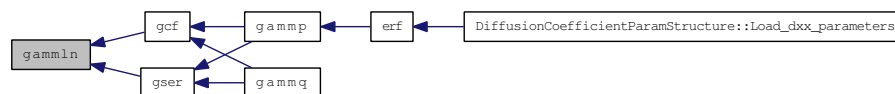
9.12.3.2 double gammln (double xx)

Returns the value $\ln[\Gamma(xx)]$ for $xx > 0$.

Definition at line 23 of file erf.cpp.

Referenced by `gcf()`, and `gser()`.

Here is the caller graph for this function:



9.12.3.3 double gammp (double a, double x)

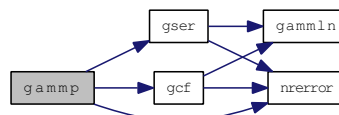
Returns the incomplete gamma function $P(a, x)$.

Definition at line 39 of file erf.cpp.

References `gcf()`, `gser()`, and `nrerror()`.

Referenced by `erf()`.

Here is the call graph for this function:



Here is the caller graph for this function:



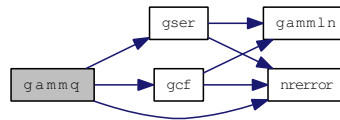
9.12.3.4 double gammq (double *a*, double *x*)

Returns the incomplete gamma function $Q(a, x) = 1 - P(a, x)$.

Definition at line 56 of file erf.cpp.

References gcf(), gser(), and nrerror().

Here is the call graph for this function:



9.12.3.5 void gcf (double * *gammcf*, double *a*, double *x*, double * *gln*)

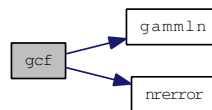
Returns the incomplete gamma function $Q(a, x)$ evaluated by its continued fraction representation as `gammcf`. Also returns $\ln(a)$ as `gln`.

Definition at line 103 of file erf.cpp.

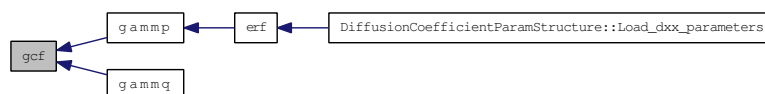
References EPS, FPMIN, gammln(), i, ITMAX, and nrerror().

Referenced by `gammp()`, and `gammq()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.3.6 void gser (double * *gamser*, double *a*, double *x*, double * *gln*)

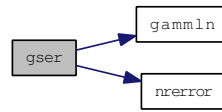
Returns the incomplete gamma function $P(a, x)$ evaluated by its series representation as `gamser`. Also returns $\ln(a)$ as `gln`.

Definition at line 74 of file erf.cpp.

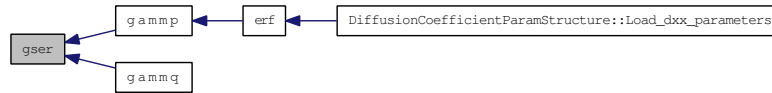
References EPS, gammln(), ITMAX, and nrerror().

Referenced by `gammp()`, and `gammq()`.

Here is the call graph for this function:



Here is the caller graph for this function:

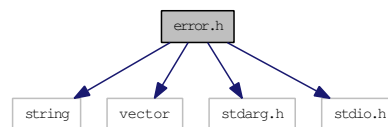


9.13 error.h File Reference

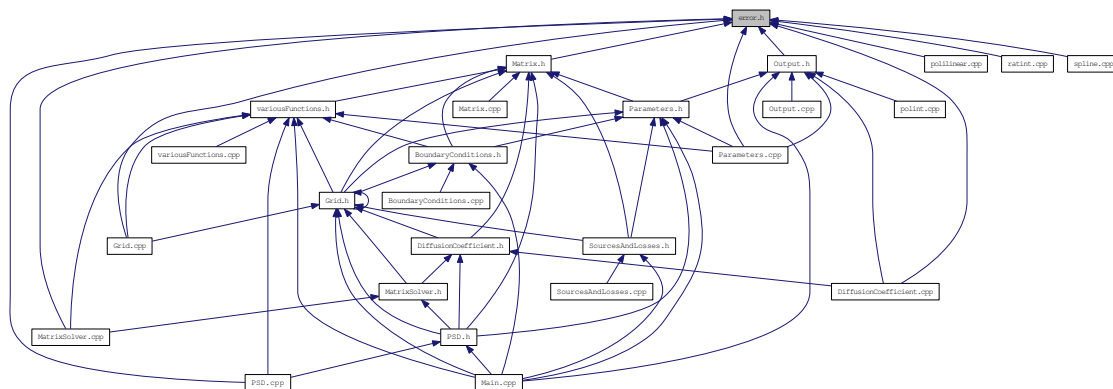
Used to work with user's exeptions.

```
#include <string>
#include <vector>
#include <stdarg.h>
#include <stdio.h>
```

Include dependency graph for error.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [single_error](#)
Hold some information about error in the code.
- class [error_msg](#)
Error message - stack of [single_error](#) s.

9.13.1 Detailed Description

Used to work with user's exeptions.

It creates stack of error messages that can be caught and outputed to the screen. Initial error is generated somewhere, and passed to called function, where mode description can be added. That is stack. Wahtever....

Todo

Remove from THE CODE error class, output class, move enclosed classes out, remove variabilities in function collings, change all enum's to strings. In that case, probably, the code can be understood.

Author:

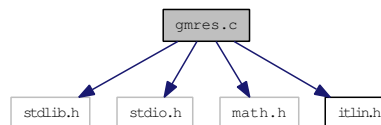
Developed under supervision of the PI Yuri Shprits

Definition in file [error.h](#).

9.14 gmres.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "itlin.h"
```

Include dependency graph for gmres.c:



Defines

- #define [MAXITER_DEFAULT](#) 100

Functions

- double [gmres_norm2](#) (int *n*, double **v*)
- int [gmres_qrfact](#) (int *n*, int *lda*, double **a*, double **q*, int *ijob*)
- void [gmres_qrsolv](#) (int *n*, int *lda*, double **a*, double **q*, double **b*)
- void [gmres](#) (int *n*, double **y*, [MATVEC](#) **matvec*, [PRECON](#) **preconr*, [PRECON](#) **preconl*, double **b*, struct [ITLIN_OPT](#) **opt*, struct [ITLIN_INFO](#) **info*)

Variables

- struct [ITLIN_IO](#) * [itlin_ioctl](#)

9.14.1 Define Documentation

9.14.1.1 #define MAXITER_DEFAULT 100

Definition at line 296 of file gmres.c.

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

9.14.2 Function Documentation

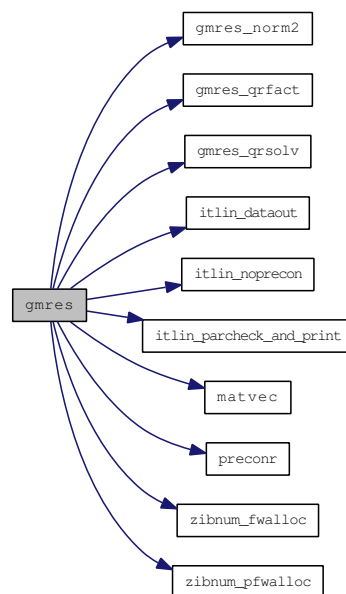
9.14.2.1 void gmres (int *n*, double **y*, [MATVEC](#) **matvec*, [PRECON](#) **preconr*, [PRECON](#) **preconl*, double **b*, struct [ITLIN_OPT](#) **opt*, struct [ITLIN_INFO](#) **info*)

Definition at line 300 of file gmres.c.

References CheckEachIter, CheckOnRestart, ITLIN_DATA::codeid, DATA, ITLIN_OPT::datafile, ITLIN_OPT::datalevel, DATALEVEL, ERROR, ITLIN_OPT::errorfile, ITLIN_OPT::errorlevel, ERRORLEVEL, False, ITLIN_DATA::Final, FITER, FMISC, FRES, ITLIN_DATA::GMRES, gmres_norm2(), gmres_qrfact(), gmres_qrsolv(), i, ITLIN_OPT::i_max, ITLIN_DATA::Initial, ITLIN_DATA::Intermediate, ITLIN_INFO::iter, ITLIN_OPT::iterfile, itlin_dataout(), itlin_noprecon(), itlin_parcheck_and_print(), matvec(), ITLIN_OPT::maxiter, MAXITER_DEFAULT, ITLIN_OPT::miscfile, ITLIN_DATA::mode, MONITOR, ITLIN_OPT::monitorfile, ITLIN_OPT::monitorlevel, MONITORLEVEL, ITLIN_INFO::nomatvec, ITLIN_INFO::noprecl, ITLIN_INFO::noprecr, ITLIN_DATA::normdx, ITLIN_INFO::precision, preconr(), RCODE, ITLIN_DATA::res, ITLIN_OPT::resfile, ITLIN_DATA::residuum, ITLIN_DATA::Solution, ITLIN_DATA::t, ITLIN_DATA::tau, ITLIN_OPT::termcheck, ITLIN_OPT::tol, True, zibnum_fwalloc(), and zibnum_pfwalloc().

Referenced by gmres_wrapout().

Here is the call graph for this function:



Here is the caller graph for this function:



9.14.2.2 double gmres_norm2 (int *n*, double * *v*)

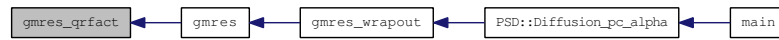
9.14.2.3 int gmres_qrfact (int *n*, int *lda*, double * *a*, double * *q*, int *ijob*)

Definition at line 502 of file gmres.c.

References i, and ITLIN_DATA::t.

Referenced by gmres().

Here is the caller graph for this function:



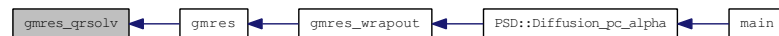
9.14.2.4 void gmres_qrsolv (int *n*, int *lda*, double * *a*, double * *q*, double * *b*)

Definition at line 547 of file `gmres.c`.

References `i`, and `ITLIN_DATA::t`.

Referenced by `gmres()`.

Here is the caller graph for this function:



9.14.3 Variable Documentation

9.14.3.1 struct ITLIN_IO* itlin_ioctl

Definition at line 194 of file `utils.c`.

9.15 gmres.d File Reference

9.16.2 Define Documentation

9.16.2.1 #define maxERR 1.e-6

Definition at line 18 of file Grid.cpp.

Referenced by Grid::MakeGrid().

9.16.3 Function Documentation

9.16.3.1 double find_alpha (double *RHS*, double *alpha_min*, double *alpha_max*, double *ERR*, int *max_it*, int *it*)

Function finds alpha-root of equation $Y(\alpha)/\sin(\alpha) = \text{RHS}$.

Root finding function, used in grid construction.

. Used in grid-bilding.

Parameters:

double *RHS* - right hand side

double *alpha_min* - min alpha value to look for root

double *alpha_max* - max alpha value to look for root

double *ERR* - max error (min accuracy)

int *max_it* - max number of iterations

int *it* - current iteration number (for recurcion)

Definition at line 545 of file Grid.cpp.

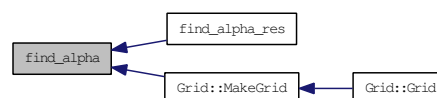
References VF::Y().

Referenced by find_alpha_res(), and Grid::MakeGrid().

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.3.2 double find_alpha_res (double *RHS*, double *alpha_min*, double *alpha_max*, double *ERR*, int *max_it*, int *it*)

Definition at line 565 of file Grid.cpp.

References find_alpha(), and VF::Y().

Here is the call graph for this function:



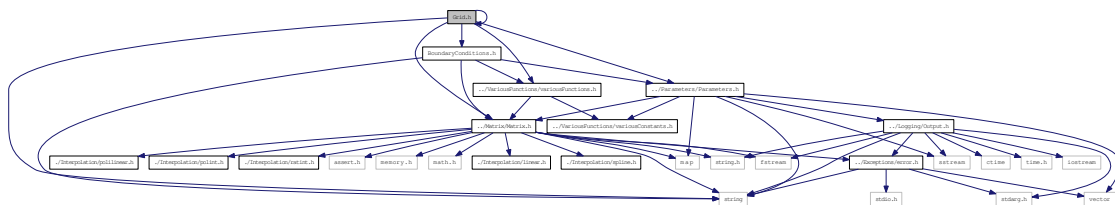
9.17 Grid.d File Reference

9.18 Grid.h File Reference

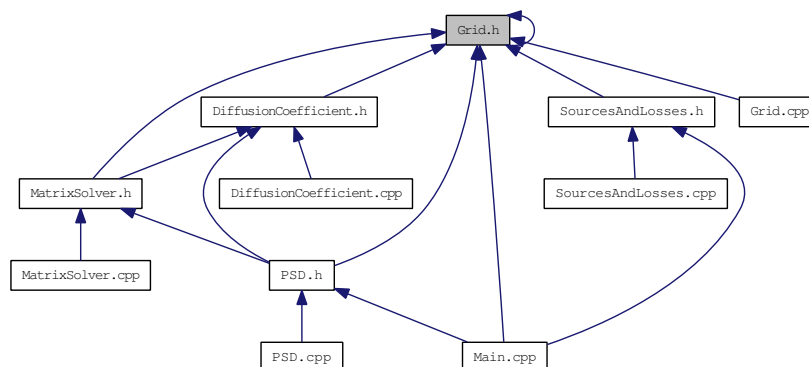
Grids, grid elements and boundary conditions headers.

```
#include <string>
#include "../Matrix/Matrix.h"
#include "../Parameters/Parameters.h"
#include "../VariousFunctions/variousFunctions.h"
#include "Grid.h"
#include "BoundaryConditions.h"
```

Include dependency graph for Grid.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [GridElement](#)
Array of values of coordinate axe.
- class [Grid](#)
Computational grid Combined from 3 grid elements: L, pc, Alpha and additional array of epc values for convenience.

Functions

- double **find_alpha** (double RHS, double alpha_min, double alhpa_max, double ERR=1e-12, int max_it=100, int it=0)

Root finding function, used in grid construction.

9.18.1 Detailed Description

Grids, grid elements and boundary conditions headers.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Grid.h](#).

9.18.2 Function Documentation

9.18.2.1 `double find_alpha (double RHS, double alpha_min, double alpha_max, double ERR, int max_it, int it)`

Root finding function, used in grid construction.

Root finding function, used in grid construction.

. Used in grid-bilding.

Parameters:

double *RHS* - right hand side

double *alpha_min* - min alpha value to look for root

double *alpha_max* - max alpha value to look for root

double *ERR* - max error (min accuracy)

int *max_it* - max number of iterations

int *it* - current iteration number (for recurcion)

Definition at line 545 of file Grid.cpp.

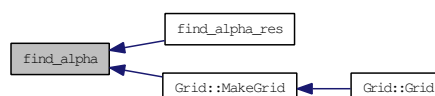
References VF::Y().

Referenced by find_alpha_res(), and Grid::MakeGrid().

Here is the call graph for this function:

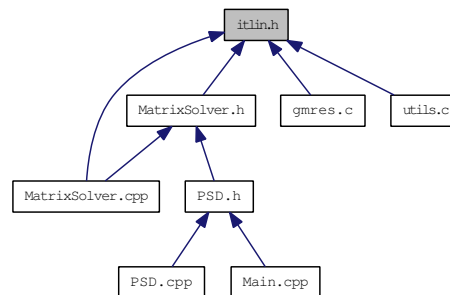


Here is the caller graph for this function:



9.19 itlin.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct [ITLIN_OPT](#)
- struct [ITLIN_INFO](#)
- struct [ITLIN_DATA](#)
- struct [ITLIN_IO](#)

Defines

- #define [RCODE](#) info → rcode
- #define [MIN](#)(A, B) (A < B ? A : B)
- #define [MAX](#)(A, B) (A > B ? A : B)
- #define [SIGN](#)(A) (A > 0 ? 1 : -1)
- #define [SMALL](#) 1.0e-150
- #define [EPMACH](#) 1.0e-17
- #define [ERRORLEVEL](#) itlin_ioctl → errlevel
- #define [MONITORLEVEL](#) itlin_ioctl → monlevel
- #define [DATALEVEL](#) itlin_ioctl → datlevel
- #define [ERROR](#) itlin_ioctl → errfile
- #define [MONITOR](#) itlin_ioctl → monfile
- #define [DATA](#) itlin_ioctl → datfile
- #define [FITER](#) itlin_ioctl → iterfile
- #define [FRES](#) itlin_ioctl → resfile
- #define [FMISC](#) itlin_ioctl → miscfile

Typedefs

- typedef void [MATVEC](#) (int n, double *x, double *b)
- typedef void [PRECON](#) (int, double *, double *)

Enumerations

- enum `PRINT_LEVEL` { `None` = 0, `Minimum` = 1, `Verbose` = 2, `Debug` = 3 }
- enum `LOGICAL` { `False` = 0, `True` = 1 }
- enum `TERM_CHECK` { `CheckOnRestart` = 0, `CheckEachIter` = 1 }
- enum `CONV_CHECK` { `Absolute` = 0, `Relative` = 1 }

Functions

- void `daxpy_` (int *n, double *alpha, double *x, int *incx, double *y, int *incy)
- int `zibnum_fwalloc` (int size, double **ptr, char vname[])
- int `zibnum_iwalloc` (int size, int **ptr, char vname[])
- int `zibnum_pfwalloc` (int size, double ***ptr, char vname[])
- double `zibnum_scaled_norm2` (int n, double *v, double *scale)
- double `zibnum_scaled_sprod` (int n, double *v1, double *v2, double *scale)
- double `zibnum_norm2` (int n, double *v)
- void `zibnum_scale` (int n, double *v1, double *v2, double *scale)
- void `zibnum_descale` (int n, double *v1, double *v2, double *scale)
- void `itlin_noprecon` (int n, double *x, double *z)
- void `itlin_dataout` (int k, int n, double *x, struct `ITLIN_DATA` *data)
- int `itlin_parcheck_and_print` (int n, `MATVEC` *matvec, struct `ITLIN_OPT` *opt, int itlin_code)

9.19.1 Define Documentation

9.19.1.1 #define DATA itlin_ioctl → datfile

Definition at line 119 of file `itlin.h`.

Referenced by `gmres()`, and `itlin_dataout()`.

9.19.1.2 #define DATALEVEL itlin_ioctl → datlevel

Definition at line 116 of file `itlin.h`.

Referenced by `gmres()`, and `itlin_dataout()`.

9.19.1.3 #define EPMACH 1.0e-17

Definition at line 58 of file `itlin.h`.

9.19.1.4 #define ERROR itlin_ioctl → errfile

Definition at line 117 of file `itlin.h`.

Referenced by `gmres()`, `itlin_parcheck_and_print()`, `zibnum_fwalloc()`, `zibnum_iwalloc()`, and `zibnum_pfwalloc()`.

9.19.1.5 #define ERRORLEVEL itlin_ioctl → errlevel

Definition at line 114 of file itlin.h.

Referenced by gmres(), itlin_parcheck_and_print(), zibnum_fwalloc(), zibnum_iwalloc(), and zibnum_pfwalloc().

9.19.1.6 #define FITER itlin_ioctl → iterfile

Definition at line 120 of file itlin.h.

Referenced by gmres(), and itlin_dataout().

9.19.1.7 #define FMISC itlin_ioctl → miscfile

Definition at line 122 of file itlin.h.

Referenced by gmres(), and itlin_dataout().

9.19.1.8 #define FRES itlin_ioctl → resfile

Definition at line 121 of file itlin.h.

Referenced by gmres(), and itlin_dataout().

9.19.1.9 #define MAX(A, B) (A > B ? A : B)

Definition at line 54 of file itlin.h.

9.19.1.10 #define MIN(A, B) (A < B ? A : B)

Definition at line 53 of file itlin.h.

Referenced by itlin_parcheck_and_print().

9.19.1.11 #define MONITOR itlin_ioctl → monfile

Definition at line 118 of file itlin.h.

Referenced by gmres(), and itlin_parcheck_and_print().

9.19.1.12 #define MONITORLEVEL itlin_ioctl → monlevel

Definition at line 115 of file itlin.h.

Referenced by gmres(), and itlin_parcheck_and_print().

9.19.1.13 #define RCODE info → rcode

Definition at line 52 of file itlin.h.

Referenced by gmres().

9.19.1.14 #define SIGN(A) (A > 0 ? 1 : -1)

Definition at line 55 of file itlin.h.

9.19.1.15 #define SMALL 1.0e-150

Definition at line 57 of file itlin.h.

Referenced by itlin_parcheck_and_print().

9.19.2 Typedef Documentation**9.19.2.1 typedef void MATVEC(int n, double *x, double *b)**

Definition at line 77 of file itlin.h.

9.19.2.2 typedef void PRECON(int, double *, double *)

Definition at line 78 of file itlin.h.

9.19.3 Enumeration Type Documentation**9.19.3.1 enum CONV_CHECK**

Enumerator:

Absolute

Relative

Definition at line 63 of file itlin.h.

9.19.3.2 enum LOGICAL

Enumerator:

False

True

Definition at line 61 of file itlin.h.

9.19.3.3 enum PRINT_LEVEL

Enumerator:

None

Minimum

Verbose

Debug

Definition at line 60 of file itlin.h.

9.19.3.4 enum TERM_CHECK

Enumerator:

CheckOnRestart

CheckEachIter

Definition at line 62 of file itlin.h.

9.19.4 Function Documentation

9.19.4.1 void daxpy_ (int * *n*, double * *alpha*, double * *x*, int * *incx*, double * *y*, int * *incy*)

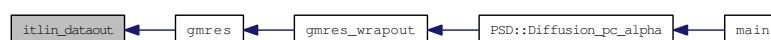
9.19.4.2 void itlin_dataout (int *k*, int *n*, double * *x*, struct ITLIN_DATA * *data*)

Definition at line 286 of file utils.c.

References ITLIN_DATA::codeid, DATA, DATALEVEL, FITER, FMISC, FRES, i, ITLIN_DATA::mode, ITLIN_DATA::normdx, ITLIN_DATA::res, ITLIN_DATA::residuum, ITLIN_DATA::t, and ITLIN_DATA::tau.

Referenced by gmres().

Here is the caller graph for this function:

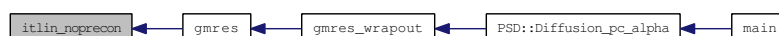


9.19.4.3 void itlin_noprecon (int *n*, double * *x*, double * *z*)

Definition at line 281 of file utils.c.

Referenced by gmres().

Here is the caller graph for this function:



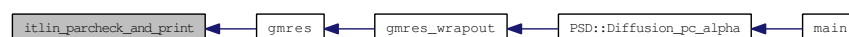
9.19.4.4 int itlin_parcheck_and_print (int *n*, MATVEC * *matvec*, struct ITLIN_OPT * *opt*, int *itlin_code*)

Definition at line 329 of file utils.c.

References Absolute, ITLIN_OPT::convcheck, ERROR, ERRORLEVEL, i, ITLIN_OPT::i_max, ITLIN_OPT::maxiter, MIN, MONITOR, MONITORLEVEL, Relative, ITLIN_OPT::rho, ITLIN_OPT::scale, SMALL, ITLIN_OPT::tol, TOLMAX, and TOLMIN.

Referenced by gmres().

Here is the caller graph for this function:



9.19.4.5 void zibnum_descale (int *n*, double * *v1*, double * *v2*, double * *scale*)

Definition at line 275 of file utils.c.

References i.

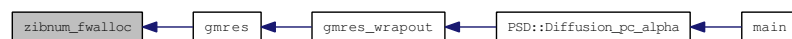
9.19.4.6 int zibnum_fwalloc (int *size*, double ** *ptr*, char *vname*[])

Definition at line 196 of file utils.c.

References ERROR, ERRORLEVEL, and i.

Referenced by gmres().

Here is the caller graph for this function:



9.19.4.7 int zibnum_iwalloc (int *size*, int ** *ptr*, char *vname*[])

Definition at line 213 of file utils.c.

References ERROR, ERRORLEVEL, and i.

9.19.4.8 double zibnum_norm2 (int *n*, double * *v*)

Definition at line 262 of file utils.c.

References i.

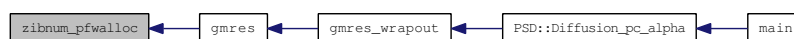
9.19.4.9 int zibnum_pfwalloc (int *size*, double *** *ptr*, char *vname*[])

Definition at line 230 of file utils.c.

References ERROR, ERRORLEVEL, and i.

Referenced by gmres().

Here is the caller graph for this function:



9.19.4.10 void zibnum_scale (int *n*, double * *v1*, double * *v2*, double * *scale*)

Definition at line 269 of file utils.c.

References i.

9.19.4.11 double zibnum_scaled_norm2 (int *n*, double * *v*, double * *scale*)

Definition at line 247 of file utils.c.

References [i](#).

9.19.4.12 double zibnum_scaled_sprod (int *n*, double * *v1*, double * *v2*, double * *scale*)

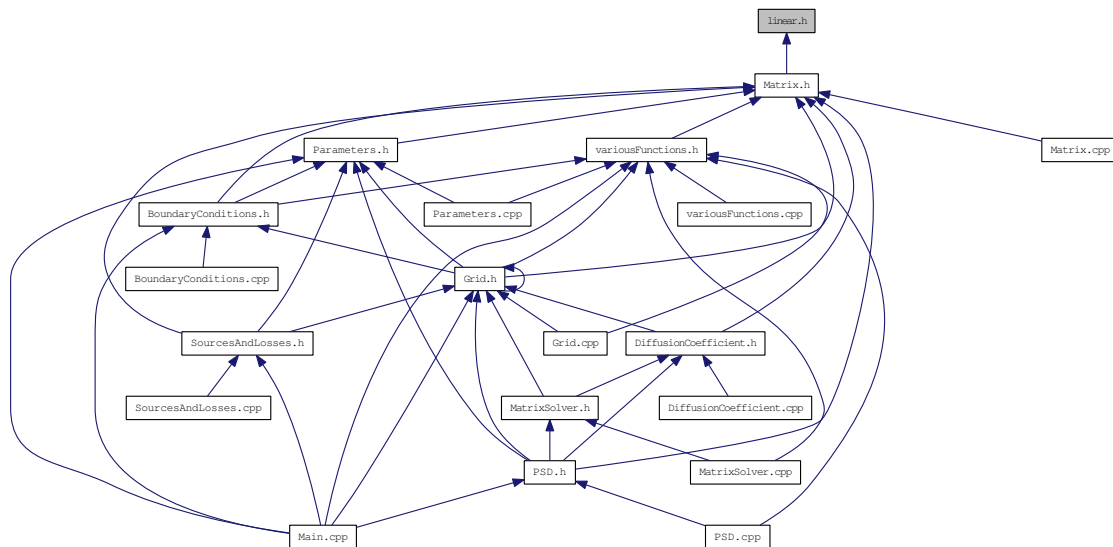
Definition at line 254 of file utils.c.

References [i](#).

9.20 linear.h File Reference

Linear interpolation.

This graph shows which files directly or indirectly include this file:



Classes

- class [Maths::Interpolation::Linear](#)
Linear interpolation.

Namespaces

- namespace [Maths](#)
- namespace [Maths::Interpolation](#)

9.20.1 Detailed Description

Linear interpolation.

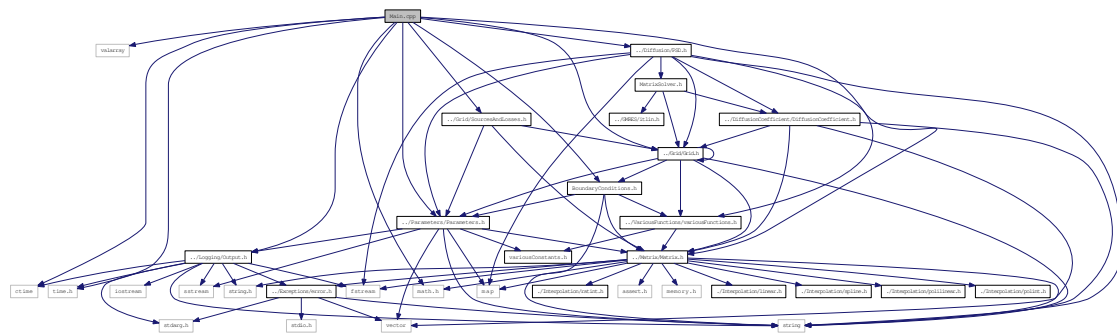
Definition in file [linear.h](#).

9.21 Main.cpp File Reference

Main program file.

```
#include <valarray>
#include <math.h>
#include <ctime>
#include <time.h>
#include "../Logging/Output.h"
#include "../VariousFunctions/variousFunctions.h"
#include "../Diffusion/PSD.h"
#include "../Grid/Grid.h"
#include "../Grid/BoundaryConditions.h"
#include "../Grid/SourcesAndLosses.h"
#include "../Parameters/Parameters.h"
```

Include dependency graph for Main.cpp:



Defines

- `#define _CRT_SECURE_NO_DEPRECATED`
- `#define VERB_VERSION_NUMBER 2.00`

Functions

- `bool check_time` (double time, double interval)
Small routine for checking if it is time for next output.
- `bool check_time` (int iter, int d_iter)
Small routine for checking if it is right iteration for the next output.
- `int main` (int argc, char *argv[])
Main code.

Variables

- [ParamStructure](#) parameters

9.21.1 Detailed Description

Main program file.

Author:

Developed under supervision of the PI Yuri Shprits

This is the main program file.

Definition in file [Main.cpp](#).

9.21.2 Define Documentation

9.21.2.1 `#define _CRT_SECURE_NO_DEPRECATED`

Definition at line 66 of file Main.cpp.

9.21.2.2 `#define VERB_VERSION_NUMBER 2.00`

Definition at line 70 of file Main.cpp.

9.21.3 Function Documentation

9.21.3.1 `bool check_time (int iter, int d_iter)`

Small routine for checking if it is right iteration for the next output.

Definition at line 109 of file Main.cpp.

9.21.3.2 `bool check_time (double time, double interval)`

Small routine for checking if it is time for next output.

Definition at line 101 of file Main.cpp.

Referenced by `main()`.

Here is the caller graph for this function:



9.21.3.3 int main (int argc, char * argv[])

Main code.

Parameters:

← *argv[1]* The name of the parameter file

Todo

Initialize memory for sources only if we have sources. Actually, I guess we should load sources each time step from a file.

Move writing to the file of 1d parameters somewhere out of main

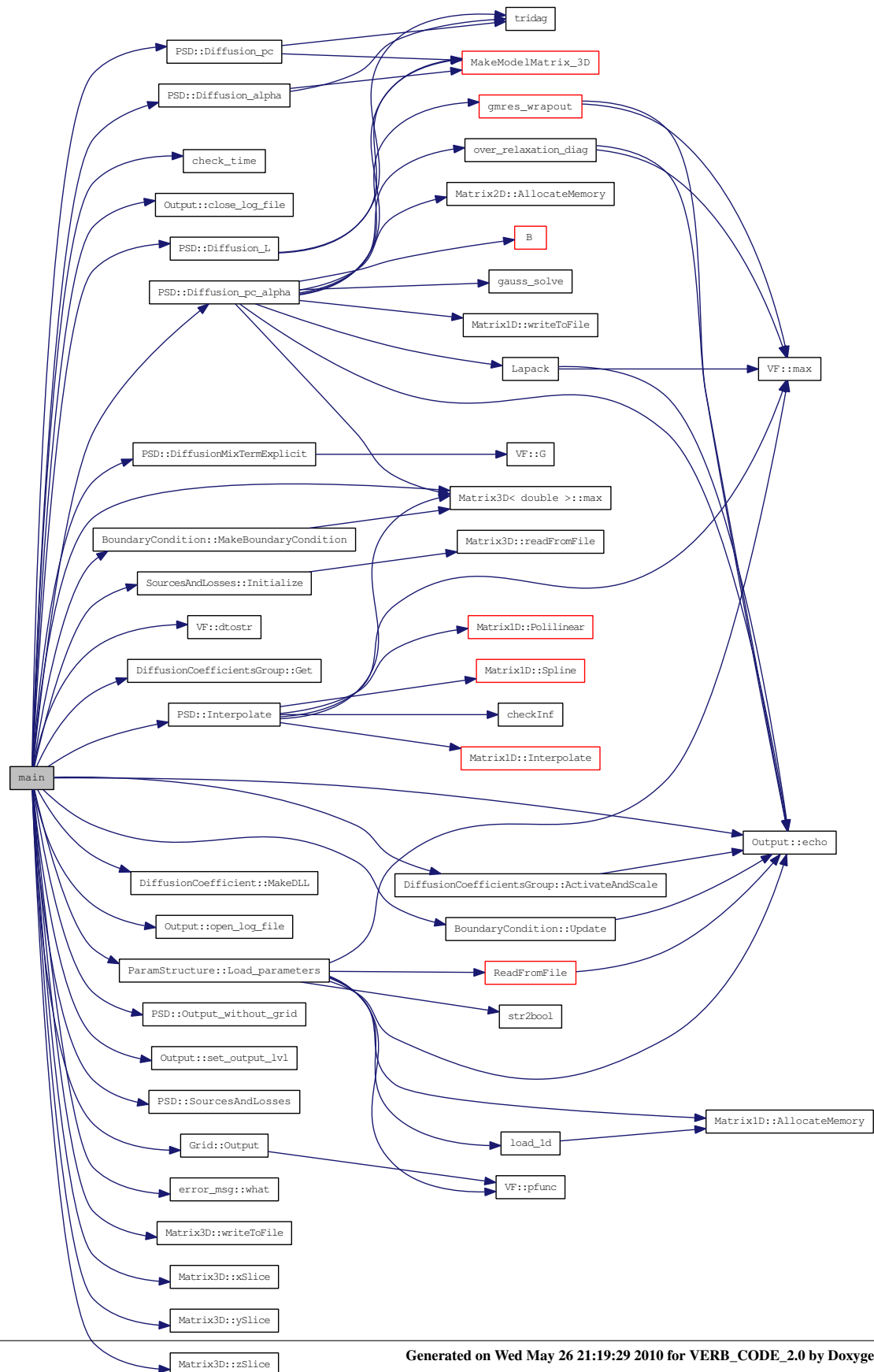
Todo

Move diffusion coefficients scaling output out of main

Definition at line 122 of file Main.cpp.

References DiffusionCoefficientsGroup::ActivateAndScale(), Grid::alpha, ParamStructure::alpha_LowerBoundaryCondition, ParamStructure::alpha_UpperBoundaryCondition, ParamStructure::PSD::approximationMethod, ParamStructure::Bf, BoundaryCondition::calculationType, check_time(), Output::close_log_file(), PSD::Diffusion_alpha(), PSD::Diffusion_L(), PSD::Diffusion_pc(), PSD::Diffusion_pc_alpha(), PSD::DiffusionMixTermExplicit(), ParamStructure::DLLType, VF::dtostr(), DiffusionCoefficientsGroup::DxxList, ParamStructure::DxxParamStructureList, Output::echo(), Grid::epc, err, ParamStructure::General_Output_parameters::fileName1D, ParamStructure::General_Output_parameters::folderName, ParamStructure::general_Output_parameters, DiffusionCoefficientsGroup::Get(), SourcesAndLosses::Initialize(), PSD::Interpolate(), ParamStructure::interpolation, ParamStructure::General_Output_parameters::iterStep, Grid::Jacobian, ParamStructure::Kp, Grid::L, ParamStructure::L_LowerBoundaryCondition, ParamStructure::L_UpperBoundaryCondition, ParamStructure::Load_parameters(), ParamStructure::localDiffusionsGrid_alpha, ParamStructure::localDiffusionsGrid_epc, ParamStructure::localDiffusionsGrid_filename, ParamStructure::localDiffusionsGrid_L, ParamStructure::localDiffusionsGrid_pc, ParamStructure::localDiffusionsGrid_type, ParamStructure::Lpp, BoundaryCondition::MakeBoundaryCondition(), DiffusionCoefficient::MakeDLL(), Matrix3D< T >::max(), Matrix3D< T >::name, ParamStructure::NoNegative, Output::open_log_file(), Grid::Output(), PSD::Output_without_grid(), ParamStructure::outputLvl, ParamStructure::outputModelMatrix, Grid::pc, ParamStructure::pc_LowerBoundaryCondition, ParamStructure::pc_UpperBoundaryCondition, ParamStructure::psdLocalDiffusions, ParamStructure::psdRadialDiffusion, ParamStructure::radialDiffusionGrid_alpha, ParamStructure::radialDiffusionGrid_epc, ParamStructure::radialDiffusionGrid_filename, ParamStructure::radialDiffusionGrid_L, ParamStructure::radialDiffusionGrid_pc, ParamStructure::radialDiffusionGrid_type, Output::set_output_lvl(), GridElement::size, ParamStructure::SL::SL_E_min, ParamStructure::SL::SL_L_top, PSD::SourcesAndLosses(), ParamStructure::sourcesAndLosses, ParamStructure::tau, ParamStructure::tauLpp, ParamStructure::timeStep, ParamStructure::totalIterationsNumber, BoundaryCondition::Update(), ParamStructure::useAlphaDifusion, ParamStructure::useEnergyAlphaMixedTerms, ParamStructure::useEnergyDifusion, ParamStructure::useRadialDifusion, error_msg::what(), Matrix3D< T >::writeToFile(), Matrix3D< T >::xSlice(), Matrix3D< T >::ySlice(), and Matrix3D< T >::zSlice().

Here is the call graph for this function:



9.21.4 Variable Documentation

9.21.4.1 ParamStructure parameters

Definition at line 118 of file Main.cpp.

9.22 Main.d File Reference

9.23 Make_boundary.m File Reference

Functions

- else `Kp_24_max` (*i*)
- `Bf` (:, 2)
- `Bf2` (:, 2)
- `Bf3` (:, 2)

Variables

- close `all`
- load `Kp` `dat`
- `Bf` = `Kp`
- `Bf1` = `Kp`
- `Bf3`
- `Kp_24_max` = `squeeze(zeros(length(Kp(:,2)),1))`
- for *i*
- end end `B_unnorm` = $10.^{((-1)*Kp(:,2)/Kp_baseline)}$

9.23.1 Function Documentation

9.23.1.1 `Bf` (:, 2)

9.23.1.2 `Bf2` (:, 2)

9.23.1.3 `Bf3` (:, 2)

9.23.1.4 else `Kp_24_max` (*i*)

9.23.2 Variable Documentation

9.23.2.1 clear `all`

Definition at line 1 of file `Make_boundary.m`.

9.23.2.2 `B_unnorm` = $10.^{((-1)*Kp(:,2)/Kp_baseline)}$

Definition at line 18 of file `Make_boundary.m`.

9.23.2.3 `Bf` = `Kp`

Definition at line 5 of file `Make_boundary.m`.

9.23.2.4 `Bf1` = `Kp`

Definition at line 6 of file `Make_boundary.m`.

9.23.2.5 Bf3

Initial value:

```
Kp
Kp_baseline=1
```

Definition at line 7 of file Make_boundary.m.

9.23.2.6 load Kp dat

Definition at line 4 of file Make_boundary.m.

9.23.2.7 for i

Initial value:

```
1:1:length(Kp(:,2))
    if (i>(23+24))
        Kp_24_max(i)=max(Kp(i-(23+24):i,2))
```

Definition at line 10 of file Make_boundary.m.

Referenced by Matrix3D< T >::AllocateMemory(), deflate(), diff_poly(), Dxx_local(), find_quad(), gcf(), get_quads(), Maths::Interpolation::Linear::getValue(), gmres(), gmres_norm2(), gmres_qrifact(), gmres_qrsolv(), Matrix2D< T >::Interpolate(), Matrix1D< T >::Interpolate(), itlin_dataout(), itlin_parcheck_and_print(), VF::J_L7_corrected(), Lapack(), Maths::Interpolation::Linear::Linear(), Linear2D(), load_1d(), PSD::Load_initial_f(), PSD::Load_initial_f_2d(), ParamStructure::Load_parameters(), MakeMatrix(), matrix(), over_relaxation_diag(), polilinear(), Matrix1D< T >::Polilinear(), polint(), Matrix1D< T >::Polint(), ratint(), Matrix1D< T >::Ratint(), rrouts(), SolveMatrix(), spline(), Matrix1D< T >::Spline(), steady_state(), error_msg::what(), zibnum_descale(), zibnum_fwalloc(), zibnum_iwalloc(), zibnum_norm2(), zibnum_pfwalloc(), zibnum_scale(), zibnum_scaled_norm2(), and zibnum_scaled_sprod().

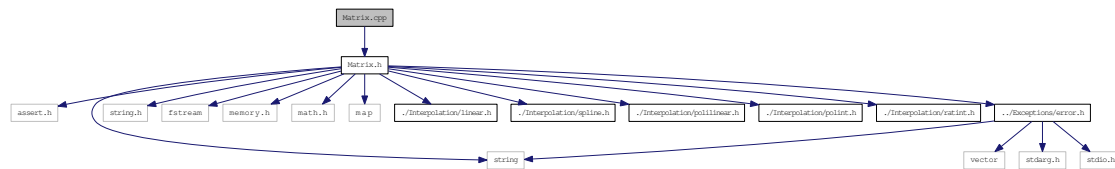
9.23.2.8 Kp_24_max = squeeze(zeros(length(Kp(:,2)),1))

Definition at line 9 of file Make_boundary.m.

9.24 Matrix.cpp File Reference

```
#include "Matrix.h"
```

Include dependency graph for Matrix.cpp:



Functions

- `template<class T >`
`T * matrix (long Rows)`
Allocating memory for 1D matrix.
- `template<class T >`
`T ** matrix (long Rows, long Columns)`
Initilizing memory for 2D matrix.
- `template<class T >`
`T *** matrix (int size_x, int size_y, int size_z)`
Initilizing memory for 3D matrix.
- `template<class T >`
`void free_matrix (T *m)`
Freing memory for 1D matrix.
- `template<class T >`
`void free_matrix (T **m)`
Freing memory for 2D matrix.
- `template<class T >`
`void free_matrix (T ***m, int size_x, int size_y)`
Freing memory for 3D matrix.
- `template<class T >`
`double Linear2D (double x, double y, T &old_grid_x, T &old_grid_y, T &f)`
Linear2D.

9.24.1 Detailed Description

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Matrix.cpp](#).

9.24.2 Function Documentation

9.24.2.1 `template<class T> void free_matrix (T *** m, int size_x, int size_y)` `[inline]`

Freing memory for 3D matrix.

Todo

Move to [Matrix.cpp](#)

Definition at line 102 of file Matrix.cpp.

9.24.2.2 `template<class T> void free_matrix (T ** m)` `[inline]`

Freing memory for 2D matrix.

Todo

Move to [Matrix.cpp](#)

Definition at line 95 of file Matrix.cpp.

9.24.2.3 `template<class T> void free_matrix (T * m)` `[inline]`

Freing memory for 1D matrix.

Todo

Move to [Matrix.cpp](#)

Definition at line 89 of file Matrix.cpp.

9.24.2.4 `template<class T> double Linear2D (double x, double y, T & old_grid_x, T & old_grid_y, T & f)` `[inline]`

Linear2D.

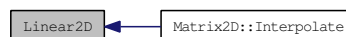
Does not work probably or works only for regular grid. Some other wired interpolation attempts.

Definition at line 1528 of file Matrix.cpp.

References i.

Referenced by `Matrix2D< T >::Interpolate()`.

Here is the caller graph for this function:



9.24.2.5 `template<class T> T*** matrix (int size_x, int size_y, int size_z)` [inline]

Initilizing memory for 3D matrix.

Todo

Move to [Matrix.cpp](#)

Definition at line 55 of file Matrix.cpp.

9.24.2.6 `template<class T> T** matrix (long Rows, long Columns)` [inline]

Initilizing memory for 2D matrix.

Todo

Move to [Matrix.cpp](#)

Definition at line 34 of file Matrix.cpp.

References [i](#).

9.24.2.7 `template<class T> T* matrix (long Rows)` [inline]

Allocating memory for 1D matrix.

Todo

Move to [Matrix.cpp](#)

Definition at line 22 of file Matrix.cpp.

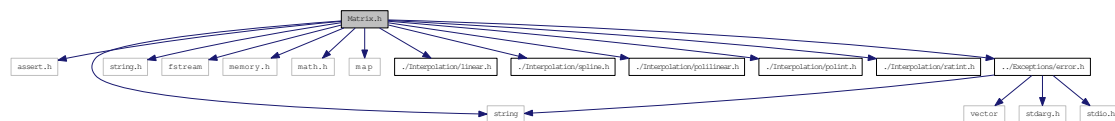
9.25 Matrix.d File Reference

9.26 Matrix.h File Reference

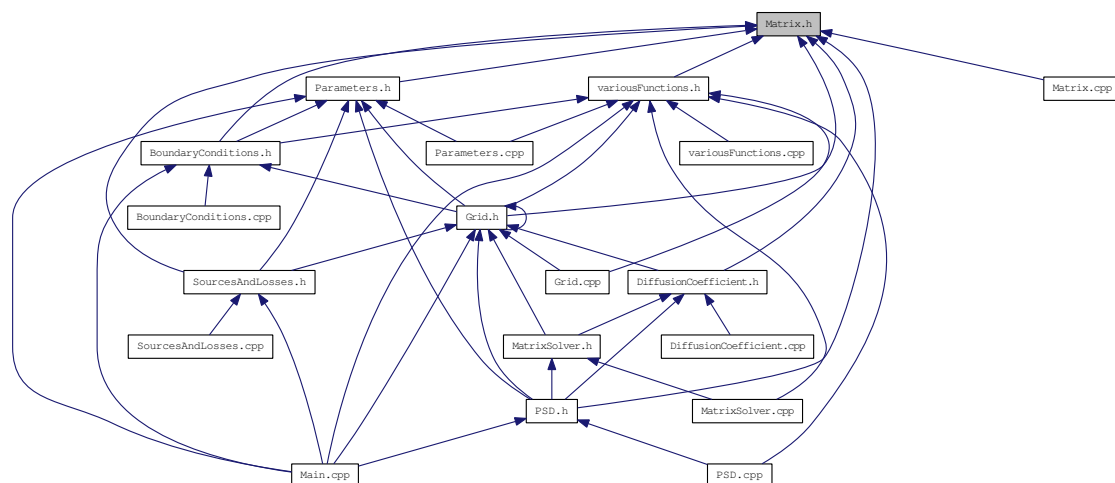
Matrix 1D, 2D and 3D and operations with them.

```
#include <assert.h>
#include <string>
#include <string.h>
#include <fstream>
#include <memory.h>
#include <math.h>
#include <map>
#include "../Interpolation/linear.h"
#include "../Interpolation/spline.h"
#include "../Interpolation/polilinear.h"
#include "../Interpolation/polint.h"
#include "../Interpolation/ratint.h"
#include "../Exceptions/error.h"
```

Include dependency graph for Matrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Matrix1D< T >](#)

Matrix 1D class.

- class [Matrix2D< T >](#)

Matrix 2D class.

- class [Matrix3D< T >](#)

Matrix 3D class.

- class [CalculationMatrix](#)

Model matrix (or related matrices) It is based on Diagonal matrix and have methods for conversion from 3D or 2D [PSD](#) (and related) arrays into 1d array of unknown elements.

Typedefs

- typedef map< int, [Matrix1D< double >](#) > [DiagMatrix](#)

Diagonal matrix.

9.26.1 Detailed Description

Matrix 1D, 2D and 3D and operations with them.

File has 1D-class, 2D-class and 3D-class of matrixes and variouse functions to work with them.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Matrix.h](#).

9.26.2 Typedef Documentation

9.26.2.1 typedef map<int , Matrix1D<double> > DiagMatrix

Diagonal matrix.

This method of storage for matrices is convenient for diagonal (spread) matrices. Stored as map (diagonal number, 1d diagonal array) The USED diagonals of the matrix are stored in 1d arrays.

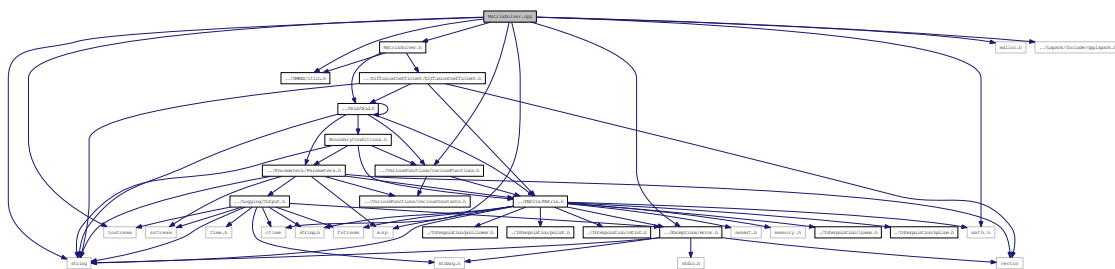
Definition at line 213 of file Matrix.h.

9.27 MatrixSolver.cpp File Reference

Making model matrixes, solving model matrixes.

```
#include "MatrixSolver.h"
#include <math.h>
#include <malloc.h>
#include <string>
#include <ctime>
#include "../VariousFunctions/variousFunctions.h"
#include "../Exceptions/error.h"
#include "../Lapack/Include/cpplapack.h"
#include "../GMRES/itlin.h"
#include <iostream>
```

Include dependency graph for MatrixSolver.cpp:



Defines

- #define EE 1.e-19
- #define I(l, k) (l)*n+(k)

Functions

- void `matvec` (int m_size, double *x, double *b)
Vector to matrix multiplication for GMRES.
 - void `preconr` (int n, double *x, double *b)
 - double `gmres_norm2` (int n, double *v)
 - void `gmres` (int n, double *y, `MATVEC` *matvec, `PRECON` *preconr, `PRECON` *preconl, double *b, struct `ITLIN_OPT` *opt, struct `ITLIN_INFO` *info)
 - bool `MakeMatrix` (double *A, double *B1, double *C, double *R, double *x, double tau, double n, double f_lower, double f_upper, int nx, double dt, double *Dxx, double taulc, double alc, int g_flag, string lower_border_condition_type, string upper_border_condition_type, string approximationMethod)
- Make matrix for 1D diffusion.*

- bool [SolveMatrix](#) (double *f, double *A, double *B1, double *C, double *R, double f_lower, double f_upper, int nx, double dt)
Solver for 1d-diffusion matrix (tridiagonal).
- void [AddBoundary](#) ([DiagMatrix](#) &Matrix_A, string type, int in, int id1, double dh)
Supportive sub-function to add boundary conditions to model matrix.
- bool [MakeModelMatrix_3D](#) ([CalculationMatrix](#) &matr_A, [CalculationMatrix](#) &matr_B, [CalculationMatrix](#) &matr_C, [Matrix3D](#)< double > &L, [Matrix3D](#)< double > &pc, [Matrix3D](#)< double > &alpha, int L_size, int pc_size, int alpha_size, [Matrix2D](#)< double > &L_lowerBoundaryCondition, [Matrix2D](#)< double > &L_upperBoundaryCondition, [Matrix2D](#)< double > &pc_lowerBoundaryCondition, [Matrix2D](#)< double > &pc_upperBoundaryCondition, [Matrix2D](#)< double > &alpha_lowerBoundaryCondition, [Matrix2D](#)< double > &alpha_upperBoundaryCondition, string L_lowerBoundaryCondition_calculationType, string L_upperBoundaryCondition_calculationType, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType, [Matrix3D](#)< double > &DLL, [Matrix3D](#)< double > &Dpcpc, [Matrix3D](#)< double > &DpcpcLpp, [Matrix3D](#)< double > &Daa, [Matrix3D](#)< double > &DaaLpp, [Matrix3D](#)< double > &Dpca, [Matrix3D](#)< double > &DpcaLpp, [Matrix3D](#)< double > &Jacobian, double dt, double Lpp, double tau, double tauLpp)
*Create matrix form of the Fokker-Planck equation: $\text{matr_A} * \text{PSD}(t+1) = \text{matr_B} * \text{PSD}(t) + \text{matr_C}$.*
- void [jacobi](#) (int m_size, double *z, double *w)
Jacobi (or diagonal) preconditioner:.
- void [SOR](#) (int m_size, double *z, double *w)
SOR preconditioner:.
- void [for_norm_A](#) (int n, double *x, double *b)
Simple preconditioner.
- void [gmres_wrapout](#) ([CalculationMatrix](#) &A, [Matrix1D](#)< double > &B, [Matrix1D](#)< double > &X, int maxiter, int i_max, double max_err)
GMRES inversion.
- void [Lapack](#) ([DiagMatrix](#) &A, [Matrix1D](#)< double > &B, [Matrix1D](#)< double > &X)
Lapack inversion.
- void [over_relaxation_diag](#) ([DiagMatrix](#) &A, [Matrix1D](#)< double > &B, [Matrix1D](#)< double > &X, int max_steps, double EE)
Over relaxation iteration method.
- void [GetDerivativeVector](#) (string derivativeType, int &dL, int &dPc, int &dAlpha)
Get change in indexes according to the derivatives direction.
- void [SecondDerivativeApproximation_3D](#) ([CalculationMatrix](#) &matr_A, int il, int im, int ia, string FirstDerivative, string SecondDerivative, [Matrix3D](#)< double > &L, [Matrix3D](#)< double > &pc, [Matrix3D](#)< double > &alpha, [Matrix3D](#)< double > &Dxx, [Matrix3D](#)< double > &Jacobian, double multiplier)
Second derivative approximation, returns coefficients to be putted into the model matrix.

- double [max2](#) (double a, double b)
Function returns maximum between a and b.
- void [mult_vect2](#) (double *af, double *vf, double *wf, int nf)
Multiplocation between vector and matrix.
- void [gauss_solve](#) (double *a, double *b, int n)
Gauss inversion.
- bool [tridag](#) (double a[], double b[], double c[], double r[], double u[], long n)
Solve the $AU=R$ system of equations, where A - tridiagonal matrix nxn with diagonals a[], b[], c[].

9.27.1 Detailed Description

Making model matrixes, solving model matrixes.

Matrix form of linear equations: $A \cdot X[t+1] = B \cdot X[t]$, where A - model matrix, B - RHS, $X[t]$ - known values of function ([PSD](#)), $X[t+1]$ - unknown values of function.

In that file there are procedures for making model matrix for 1d-diffusion, 2d-diffusion, some ideas of 3d-diffusion and mixed terms. Solver for tridiagonal matrix, solver by gauss method and iteration method (upper relaxation).

This file is under development, and has a lot of commented code, unfinished etc code.

There is a checked version for 1d diffusion (or split method of 2d, 3d diffusions) - 1d_universal_solver.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [MatrixSolver.cpp](#).

9.27.2 Define Documentation

9.27.2.1 #define EE 1.e-19

Definition at line 1010 of file MatrixSolver.cpp.

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

9.27.2.2 #define I(l, k) (l)*n+(k)

Definition at line 1011 of file MatrixSolver.cpp.

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

9.27.3 Function Documentation

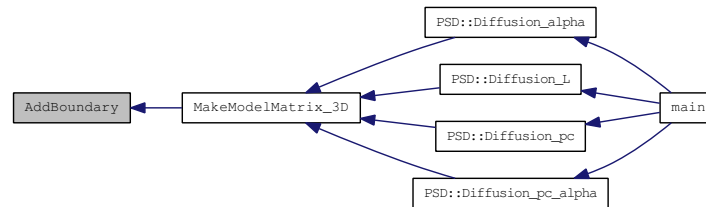
9.27.3.1 void AddBoundary (DiagMatrix & Matrix_A, string type, int in, int id1, double dh)

Supportive sub-function to add boundary conditions to model matrix.

Definition at line 267 of file MatrixSolver.cpp.

Referenced by MakeModelMatrix_3D().

Here is the caller graph for this function:



9.27.3.2 void for_norm_A (int *n*, double * *x*, double * *b*)

Simple preconditioner.

Definition at line 643 of file MatrixSolver.cpp.

9.27.3.3 void gauss_solve (double * *a*, double * *b*, int *n*)

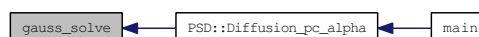
Gauss inversion.

Definition at line 1030 of file MatrixSolver.cpp.

References EE, and I.

Referenced by PSD::Diffusion_pc_alpha().

Here is the caller graph for this function:



9.27.3.4 void GetDerivativeVector (string *derivativeType*, int & *dL*, int & *dPc*, int & *dAlpha*)

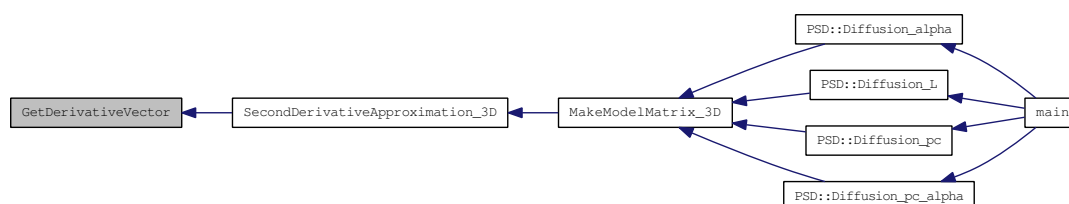
Get change in indexes according to the derivatives direction.

Used in approximation. If it gets `derivativeType == DT_ALPHA_left`, for example, that means we have alpha-left-derivative, which is $(f(\alpha) - f(\alpha-1)) / (\text{delta } \alpha)$. So it returns `dAlpha = -1`, as a derivative direction vector.

Definition at line 933 of file MatrixSolver.cpp.

Referenced by SecondDerivativeApproximation_3D().

Here is the caller graph for this function:



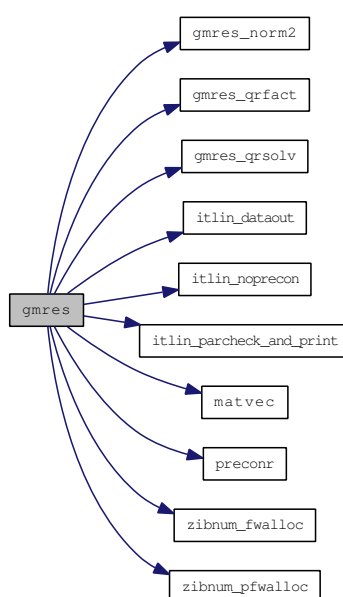
9.27.3.5 void gmres (int *n*, double * *y*, MATVEC * *matvec*, PRECON * *preconr*, PRECON * *preconl*, double * *b*, struct ITLIN_OPT * *opt*, struct ITLIN_INFO * *info*)

Definition at line 300 of file gmres.c.

References CheckEachIter, CheckOnRestart, ITLIN_DATA::codeid, DATA, ITLIN_OPT::datafile, ITLIN_OPT::datalevel, DATALEVEL, ERROR, ITLIN_OPT::errorfile, ITLIN_OPT::errorlevel, ERRORLEVEL, False, ITLIN_DATA::Final, FITER, FMISC, FRES, ITLIN_DATA::GMRES, gmres_norm2(), gmres_qrfact(), gmres_qrsolv(), i, ITLIN_OPT::i_max, ITLIN_DATA::Initial, ITLIN_DATA::Intermediate, ITLIN_INFO::iter, ITLIN_OPT::iterfile, itlin_dataout(), itlin_noprecon(), itlin_parcheck_and_print(), matvec(), ITLIN_OPT::maxiter, MAXITER_DEFAULT, ITLIN_OPT::miscfile, ITLIN_DATA::mode, MONITOR, ITLIN_OPT::monitorfile, ITLIN_OPT::monitorlevel, MONITORLEVEL, ITLIN_INFO::nomatvec, ITLIN_INFO::noprecl, ITLIN_INFO::noprecr, ITLIN_DATA::normdx, ITLIN_INFO::precision, preconr(), RCODE, ITLIN_DATA::res, ITLIN_OPT::resfile, ITLIN_DATA::residuum, ITLIN_DATA::Solution, ITLIN_DATA::t, ITLIN_DATA::tau, ITLIN_OPT::termcheck, ITLIN_OPT::tol, True, zibnum_fwalloc(), and zibnum_pfwalloc().

Referenced by gmres_wrapout().

Here is the call graph for this function:



Here is the caller graph for this function:



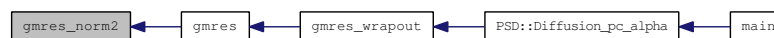
9.27.3.6 double gmres_norm2 (int *n*, double * *v*)

Definition at line 495 of file gmres.c.

References [i](#).

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

Here is the caller graph for this function:



9.27.3.7 void gmres_wrapout (CalculationMatrix & *A*, Matrix1D< double > & *B*, Matrix1D< double > & *X*, int *maxiter*, int *i_max*, double *max_err*)

GMRES inversion.

Potentially with bugs.

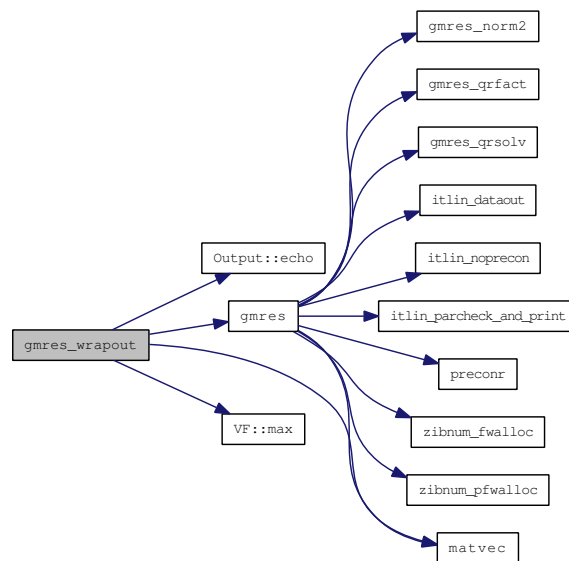
$A * X = B$ - equation

Definition at line 656 of file MatrixSolver.cpp.

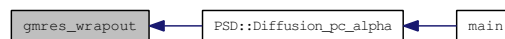
References [CheckOnRestart](#), [ITLIN_OPT::datafile](#), [ITLIN_OPT::datalevel](#), [Output::echo\(\)](#), [ITLIN_OPT::errorfile](#), [ITLIN_OPT::errorlevel](#), [gmres\(\)](#), [ITLIN_OPT::i_max](#), [ITLIN_INFO::iter](#), [ITLIN_OPT::iterfile](#), [matvec\(\)](#), [VF::max\(\)](#), [ITLIN_OPT::maxiter](#), [ITLIN_OPT::miscfile](#), [ITLIN_OPT::monitorfile](#), [ITLIN_OPT::monitorlevel](#), [ITLIN_INFO::nomatvec](#), [None](#), [ITLIN_INFO::noprecl](#), [ITLIN_INFO::noprecr](#), [ITLIN_INFO::precision](#), [ITLIN_INFO::rcode](#), [ITLIN_OPT::resfile](#), [ITLIN_OPT::termcheck](#), [ITLIN_OPT::tol](#), and [Verbose](#).

Referenced by [PSD::Diffusion_pc_alpha\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.8 void jacobi (int *m_size*, double * *z*, double * *w*)

Jacobi (or diagonal) preconditioner:.

Definition at line 601 of file MatrixSolver.cpp.

9.27.3.9 void Lapack (DiagMatrix & *A*, Matrix1D< double > & *B*, Matrix1D< double > & *X*)

Lapack inversion.

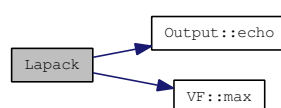
$A * X = B$ - equation

Definition at line 791 of file MatrixSolver.cpp.

References Output::echo(), i, and VF::max().

Referenced by PSD::Diffusion_pc_alpha().

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.10 `bool MakeMatrix (double * A, double * B1, double * C, double * R, double * x, double tau, double n, double f_lower, double f_upper, int nx, double dt, double * Dxx, double taulc, double alc, int g_flag, string lower_border_condition_type, string upper_border_condition_type, string approximationMethod)`

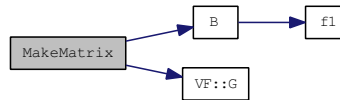
Make matrix for 1D diffusion.

Make matrix for 1d diffusion.

Definition at line 56 of file MatrixSolver.cpp.

References B(), VF::G(), and i.

Here is the call graph for this function:



9.27.3.11 `bool MakeModelMatrix_3D (CalculationMatrix & matr_A, CalculationMatrix & matr_B, CalculationMatrix & matr_C, Matrix3D< double > & L, Matrix3D< double > & pc, Matrix3D< double > & alpha, int L_size, int pc_size, int alpha_size, Matrix2D< double > & L_lowerBoundaryCondition, Matrix2D< double > & L_upperBoundaryCondition, Matrix2D< double > & pc_lowerBoundaryCondition, Matrix2D< double > & pc_upperBoundaryCondition, Matrix2D< double > & alpha_lowerBoundaryCondition, Matrix2D< double > & alpha_upperBoundaryCondition, string L_lowerBoundaryCondition_calculationType, string L_upperBoundaryCondition_calculationType, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType, Matrix3D< double > & DLL, Matrix3D< double > & Dpcpc, Matrix3D< double > & DpcpcLpp, Matrix3D< double > & Daa, Matrix3D< double > & DaaLpp, Matrix3D< double > & Dpca, Matrix3D< double > & DpcaLpp, Matrix3D< double > & Jacobian, double dt, double Lpp, double tau, double tauLpp)`

Create matrix form of the Fokker-Planck equation: $\text{matr_A} * \text{PSD}(t+1) = \text{matr_B} * \text{PSD}(t) + \text{matr_C}$.

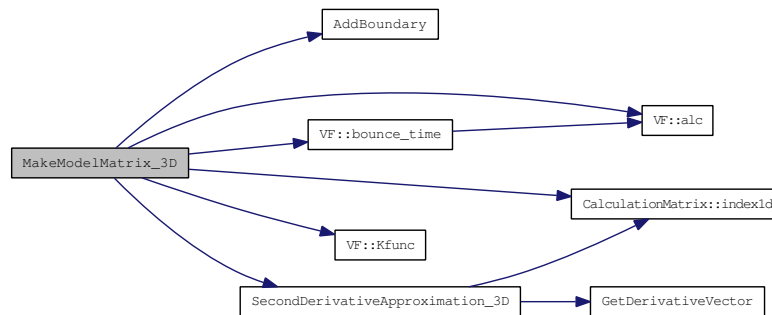
Calculation matr_A, matr_B, and matr_C, i.e. numerical approximation of the derivatives Takes boundary conditions and diffusion coefficients as an input. L, pc, alpha should be orthogonal for 3D!!!

Definition at line 287 of file MatrixSolver.cpp.

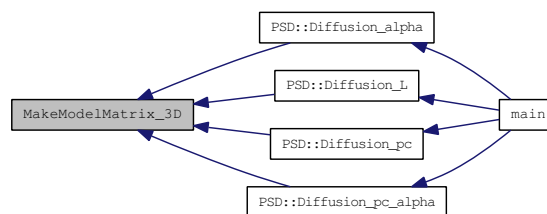
References AddBoundary(), VF::alc(), VF::bounce_time(), CalculationMatrix::change_ind, CalculationMatrix::index1d(), VF::Kfunc(), and SecondDerivativeApproximation_3D().

Referenced by PSD::Diffusion_alpha(), PSD::Diffusion_L(), PSD::Diffusion_pc(), and PSD::Diffusion_pc_alpha().

Here is the call graph for this function:



Here is the caller graph for this function:



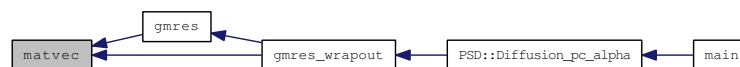
9.27.3.12 void matvec (int *n*, double * *x*, double * *b*)

Vector to matrix multiplication for GMRES.

Definition at line 559 of file MatrixSolver.cpp.

Referenced by gmres(), and gmres_wrapout().

Here is the caller graph for this function:



9.27.3.13 double max2 (double *a*, double *b*)

Function returns maximum between a and b.

Definition at line 1014 of file MatrixSolver.cpp.

9.27.3.14 void mult_vect2 (double * *af*, double * *vf*, double * *wf*, int *nf*)

Multiplocation between vector and matrix.

Definition at line 1018 of file MatrixSolver.cpp.

9.27.3.15 void over_relaxation_diag (DiagMatrix & A, Matrix1D< double > & B, Matrix1D< double > & X, int max_steps, double EE)

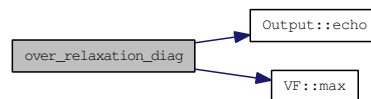
Over relaxation iteration method.

Definition at line 884 of file MatrixSolver.cpp.

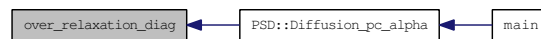
References Output::echo(), i, and VF::max().

Referenced by PSD::Diffusion_pc_alpha().

Here is the call graph for this function:



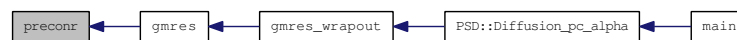
Here is the caller graph for this function:



9.27.3.16 void preconr (int n, double * x, double * b)

Referenced by gmres().

Here is the caller graph for this function:



9.27.3.17 void SecondDerivativeApproximation_3D (CalculationMatrix & matr_A, int il, int im, int ia, string FirstDerivative, string SecondDerivative, Matrix3D< double > & L, Matrix3D< double > & pc, Matrix3D< double > & alpha, Matrix3D< double > & Dxx, Matrix3D< double > & Jacobian, double multiplier)

Second derivative approximation, returns coefficients to be putted into the model matrix.

$$L_{\alpha\beta}(y) = (D_{\alpha\beta} \cdot y_{\bar{x}_\alpha})_{x_\beta}$$

Samarskiy, page 261

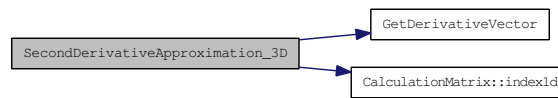
Returns coefficients to be put into model matrix for an approximation of a second derivative.

Definition at line 960 of file MatrixSolver.cpp.

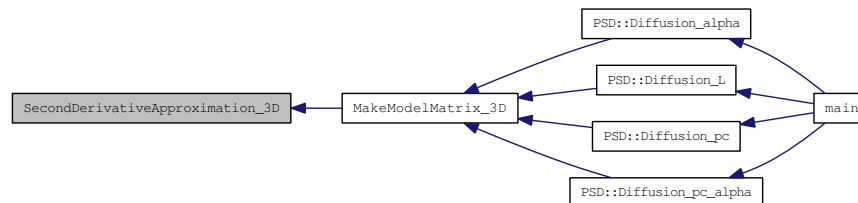
References GetDerivativeVector(), and CalculationMatrix::index1d().

Referenced by MakeModelMatrix_3D().

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.18 bool SolveMatrix (double **f*, double **A*, double **B1*, double **C*, double **R*, double *f_lower*, double *f_upper*, int *nx*, double *dt*)

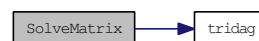
Solver for 1d-diffusion matrix (tridiagonal).

Solver for 1d general equation.

Definition at line 226 of file MatrixSolver.cpp.

References `i`, and `tridag()`.

Here is the call graph for this function:



9.27.3.19 void SOR (int *m_size*, double **z*, double **w*)

SOR preconditioner:.

$P = (D/\text{rel} + L) w = P^{-1} * z$; $P * w = z$ $(D/\text{rel} + L) * w = z$

Definition at line 621 of file MatrixSolver.cpp.

9.27.3.20 bool tridag (double *a*[], double *b*[], double *c*[], double *r*[], double *u*[], long *n*)

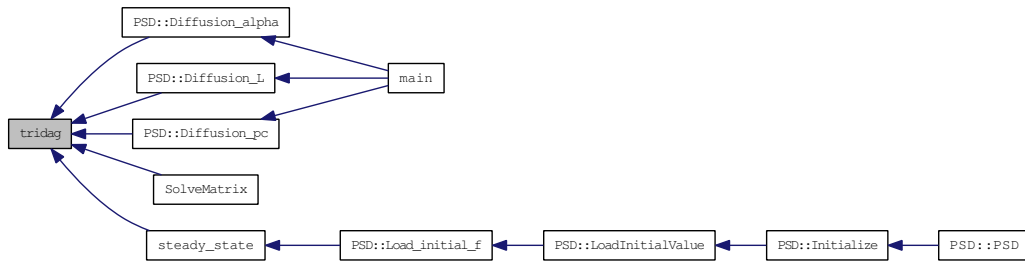
Solve the $AU=R$ system of equations, where A - tridiagonal matrix $n \times n$ with diagonals $a[]$, $b[]$, $c[]$.

Solver for a system of equations with 3-diagonal matrix.

Definition at line 1080 of file MatrixSolver.cpp.

Referenced by `PSD::Diffusion_alpha()`, `PSD::Diffusion_L()`, `PSD::Diffusion_pc()`, `SolveMatrix()`, and `steady_state()`.

Here is the caller graph for this function:



9.28 MatrixSolver.d File Reference

9.29 MatrixSolver.h File Reference

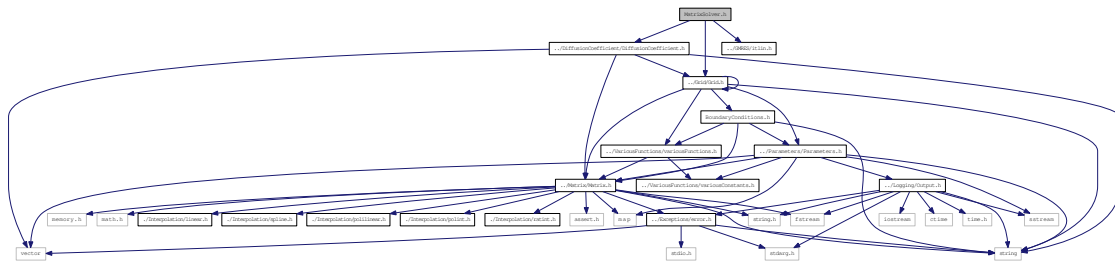
Making model matrixes, solving model matrixes.

```
#include "../Grid/Grid.h"
```

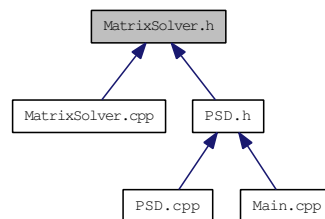
```
#include "../DiffusionCoefficient/DiffusionCoefficient.h"
```

```
#include "../GMRES/itlin.h"
```

Include dependency graph for MatrixSolver.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool [MakeMatrix](#) (double *A, double *B1, double *C, double *R, double *x, double tau, double n, double f_lower, double f_upper, int nx, double dt, double *Dxx, double taulc, double alc, int g_flag, string lower_border_condition_type, string upper_border_condition_type, string approximationMethod="AM_Split_C")

Make matrix for 1d diffusion.

- bool [SolveMatrix](#) (double *f, double *A, double *B1, double *C, double *R, double f_lower, double f_upper, int nx, double dt)

Solver for 1d general equation.

- void [over_relaxation_diag](#) (DiagMatrix &A, Matrix1D< double > &B, Matrix1D< double > &X, int max_steps=1e5, double EE=1e-3)

Over relaxation iteration method.

- void [Lapack](#) (DiagMatrix &A, Matrix1D< double > &B, Matrix1D< double > &X)

Lapack inversion.

- void [gauss_solve](#) (double *a, double *b, int n)

Gauss inversion.

- bool [tridag](#) (double a[], double b[], double c[], double r[], double u[], long n)

Solver for a system of equations with 3-diagonal matrix.

- void [gmres_wrapout](#) ([CalculationMatrix](#) &A, [Matrix1D](#)< double > &B, [Matrix1D](#)< double > &X, int maxiter=100, int i_max=10, double tol=1e-10)

GMRES inversion.

- bool [MakeModelMatrix_3D](#) ([CalculationMatrix](#) &matr_A, [CalculationMatrix](#) &matr_B, [CalculationMatrix](#) &matr_C, [Matrix3D](#)< double > &L, [Matrix3D](#)< double > &pc, [Matrix3D](#)< double > &alpha, int L_size, int pc_size, int alpha_size, [Matrix2D](#)< double > &L_lowerBoundaryCondition, [Matrix2D](#)< double > &L_upperBoundaryCondition, [Matrix2D](#)< double > &pc_lowerBoundaryCondition, [Matrix2D](#)< double > &pc_upperBoundaryCondition, [Matrix2D](#)< double > &alpha_lowerBoundaryCondition, [Matrix2D](#)< double > &alpha_upperBoundaryCondition, string L_lowerBoundaryCondition_calculationType, string L_upperBoundaryCondition_calculationType, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType, [Matrix3D](#)< double > &DLL, [Matrix3D](#)< double > &Dpcpc, [Matrix3D](#)< double > &DpcpcLpp, [Matrix3D](#)< double > &Daa, [Matrix3D](#)< double > &DaaLpp, [Matrix3D](#)< double > &Dpca, [Matrix3D](#)< double > &DpcaLpp, [Matrix3D](#)< double > &Jacobian, double dt, double Lpp, double tau=1e99, double tauLpp=1e99)

*Create matrix form of the Fokker-Planck equation: $\text{matr_A} * \text{PSD}(t+1) = \text{matr_B} * \text{PSD}(t) + \text{matr_C}$.*

- void [SecondDerivativeApproximation_3D](#) ([CalculationMatrix](#) &matr_A, int il, int im, int ia, string FirstDerivative, string SecondDerivative, [Matrix3D](#)< double > &L, [Matrix3D](#)< double > &pc, [Matrix3D](#)< double > &alpha, [Matrix3D](#)< double > &Dxx, [Matrix3D](#)< double > &Jacobian, double multiplicator)

Second derivative approximation, returns coefficients to be putted into the model matrix.

9.29.1 Detailed Description

Making model matrixes, solving model matrixes.

Matrix form of linear equations: $A * X[t+1] = B * X[t]$, where A - model matrix, B - RHS, $X[t]$ - known values of function ([PSD](#)), $X[t+1]$ - unknown values of function.

In that file there are procedures for making model matrix for 1d-diffusion, 2d-diffusion, some ideas of 3d-diffusion and mixed terms. Solver for tridiagonal matrix, solver by gauss method and iteration method (upper relaxation).

This file is under development, and has a lot of commented code, unfinished etc code.

There is a checked version for 1d diffusion (or split method of 2d, 3d diffusions) - [1d_universal_solver](#).

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [MatrixSolver.h](#).

9.29.2 Function Documentation

9.29.2.1 void gauss_solve (double * *a*, double * *b*, int *n*)

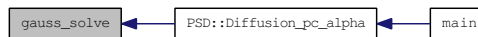
Gauss inversion.

Definition at line 1030 of file MatrixSolver.cpp.

References EE, and I.

Referenced by PSD::Diffusion_pc_alpha().

Here is the caller graph for this function:



9.29.2.2 void gmres_wrapout (CalculationMatrix & *A*, Matrix1D< double > & *B*, Matrix1D< double > & *X*, int *maxiter*, int *i_max*, double *max_err*)

GMRES inversion.

Potentially with bugs.

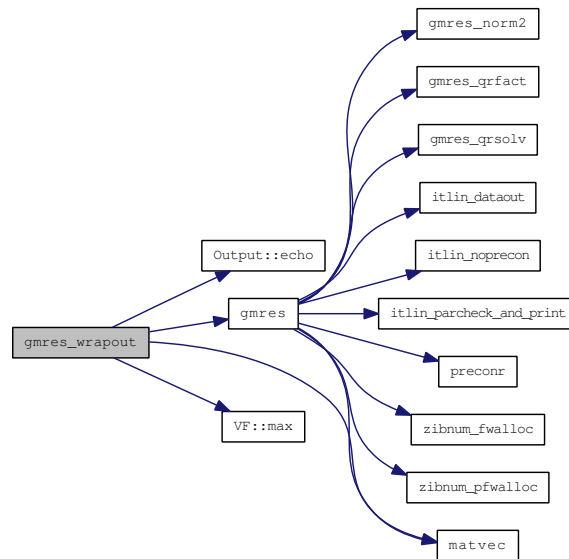
$A * X = B$ - equation

Definition at line 656 of file MatrixSolver.cpp.

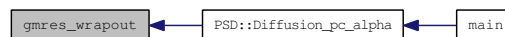
References CheckOnRestart, ITLIN_OPT::datafile, ITLIN_OPT::datalevel, Output::echo(), ITLIN_OPT::errorfile, ITLIN_OPT::errorlevel, gmres(), ITLIN_OPT::i_max, ITLIN_INFO::iter, ITLIN_OPT::iterfile, matvec(), VF::max(), ITLIN_OPT::maxiter, ITLIN_OPT::miscfile, ITLIN_OPT::monitorfile, ITLIN_OPT::monitorlevel, ITLIN_INFO::nomatvec, None, ITLIN_INFO::noprecl, ITLIN_INFO::noprecr, ITLIN_INFO::precision, ITLIN_INFO::rcode, ITLIN_OPT::resfile, ITLIN_OPT::termcheck, ITLIN_OPT::tol, and Verbose.

Referenced by PSD::Diffusion_pc_alpha().

Here is the call graph for this function:



Here is the caller graph for this function:



9.29.2.3 void Lapack (DiagMatrix & A, Matrix1D< double > & B, Matrix1D< double > & X)

Lapack inversion.

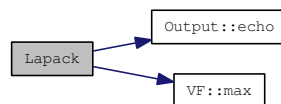
$A * X = B$ - equation

Definition at line 791 of file MatrixSolver.cpp.

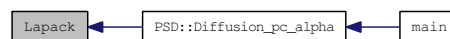
References `Output::echo()`, `i`, and `VF::max()`.

Referenced by `PSD::Diffusion_pc_alpha()`.

Here is the call graph for this function:



Here is the caller graph for this function:



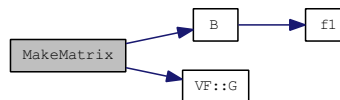
9.29.2.4 `bool MakeMatrix (double * A, double * B1, double * C, double * R, double * x, double tau, double n, double f_lower, double f_upper, int nx, double dt, double * Dxx, double taulc, double alc, int g_flag, string lower_border_condition_type, string upper_border_condition_type, string approximationMethod)`

Make matrix for 1d diffusion.

Definition at line 56 of file MatrixSolver.cpp.

References B(), VF::G(), and i.

Here is the call graph for this function:



9.29.2.5 `bool MakeModelMatrix_3D (CalculationMatrix & matr_A, CalculationMatrix & matr_B, CalculationMatrix & matr_C, Matrix3D< double > & L, Matrix3D< double > & pc, Matrix3D< double > & alpha, int L_size, int pc_size, int alpha_size, Matrix2D< double > & L_lowerBoundaryCondition, Matrix2D< double > & L_upperBoundaryCondition, Matrix2D< double > & pc_lowerBoundaryCondition, Matrix2D< double > & pc_upperBoundaryCondition, Matrix2D< double > & alpha_lowerBoundaryCondition, Matrix2D< double > & alpha_upperBoundaryCondition, string L_lowerBoundaryCondition_calculationType, string L_upperBoundaryCondition_calculationType, string pc_lowerBoundaryCondition_calculationType, string pc_upperBoundaryCondition_calculationType, string alpha_lowerBoundaryCondition_calculationType, string alpha_upperBoundaryCondition_calculationType, Matrix3D< double > & DLL, Matrix3D< double > & Dpcpc, Matrix3D< double > & DpcpcLpp, Matrix3D< double > & Daa, Matrix3D< double > & DaaLpp, Matrix3D< double > & Dpca, Matrix3D< double > & DpcaLpp, Matrix3D< double > & Jacobian, double dt, double Lpp, double tau, double tauLpp)`

Create matrix form of the Fokker-Planck equation: $\text{matr_A} * \text{PSD}(t+1) = \text{matr_B} * \text{PSD}(t) + \text{matr_C}$.

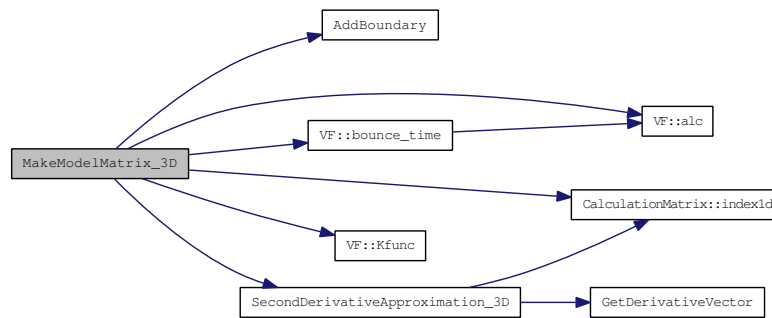
Calculation matr_A, matr_B, and matr_C, i.e. numerical approximation of the derivatives Takes boundary conditions and diffusion coefficients as an input. L, pc, alpha should be orthogonal for 3D!!!

Definition at line 287 of file MatrixSolver.cpp.

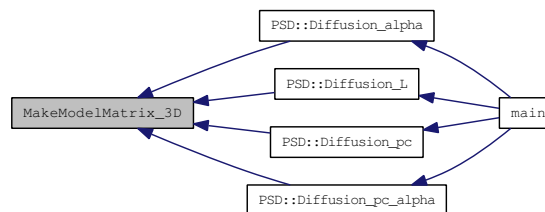
References AddBoundary(), VF::alc(), VF::bounce_time(), CalculationMatrix::change_ind, CalculationMatrix::index1d(), VF::Kfunc(), and SecondDerivativeApproximation_3D().

Referenced by PSD::Diffusion_alpha(), PSD::Diffusion_L(), PSD::Diffusion_pc(), and PSD::Diffusion_pc_alpha().

Here is the call graph for this function:



Here is the caller graph for this function:



9.29.2.6 void over_relaxation_diag (DiagMatrix & A, Matrix1D< double > & B, Matrix1D< double > & X, int max_steps = 1e5, double EE = 1e-3)

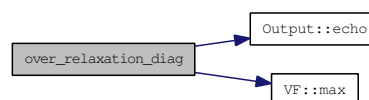
Over relaxation iteration method.

Definition at line 884 of file MatrixSolver.cpp.

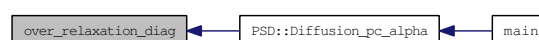
References Output::echo(), i, and VF::max().

Referenced by PSD::Diffusion_pc_alpha().

Here is the call graph for this function:



Here is the caller graph for this function:



9.29.2.7 void SecondDerivativeApproximation_3D (CalculationMatrix & *matr_A*, int *il*, int *im*, int *ia*, string *FirstDerivative*, string *SecondDerivative*, Matrix3D< double > & *L*, Matrix3D< double > & *pc*, Matrix3D< double > & *alpha*, Matrix3D< double > & *Dxx*, Matrix3D< double > & *Jacobian*, double *multiplier*)

Second derivative approximation, returns coefficients to be putted into the model matrix.

$$L_{\alpha\beta}(y) = (D_{\alpha\beta} \cdot y_{\bar{x}_\alpha})_{x_\beta}$$

Samarskiy, page 261

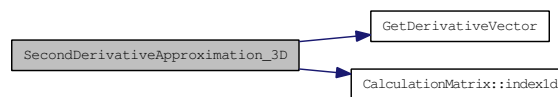
Returns coefficients to be put into model matrix for an approximation of a second derivative.

Definition at line 960 of file MatrixSolver.cpp.

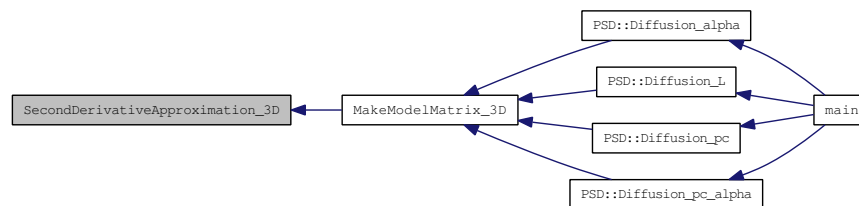
References GetDerivativeVector(), and CalculationMatrix::index1d().

Referenced by MakeModelMatrix_3D().

Here is the call graph for this function:



Here is the caller graph for this function:



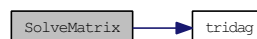
9.29.2.8 bool SolveMatrix (double **f*, double **A*, double **BI*, double **C*, double **R*, double *f_lower*, double *f_upper*, int *nx*, double *dt*)

Solver for 1d general equation.

Definition at line 226 of file MatrixSolver.cpp.

References i, and tridag().

Here is the call graph for this function:



9.29.2.9 bool tridag (double *a*[], double *b*[], double *c*[], double *r*[], double *u*[], long *n*)

Solver for a system of equations with 3-diagonal matrix.

$Au = r$ Where $A = \text{diag}(a, b, c)$,

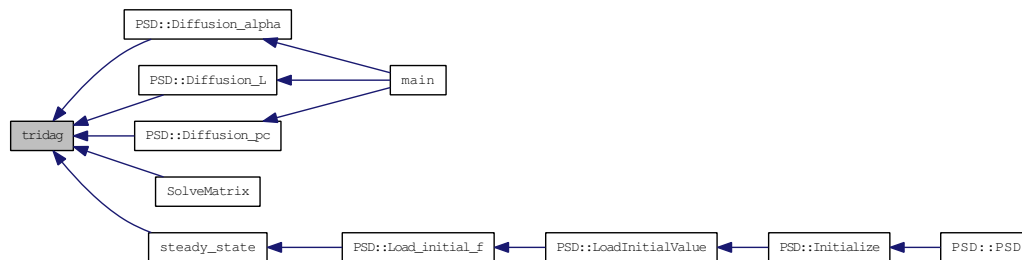
Parameters:

$a[]$ - array, diagonal '-1' of the matrix
 $b[]$ - array, diagonal '0' of the matrix
 $c[]$ - array, diagonal '+1' of the matrix
 $r[]$ - array, r-vector
 $u[]$ - array, result
 n - size of the matrix

Definition at line 1080 of file MatrixSolver.cpp.

Referenced by PSD::Diffusion_alpha(), PSD::Diffusion_L(), PSD::Diffusion_pc(), SolveMatrix(), and steady_state().

Here is the caller graph for this function:

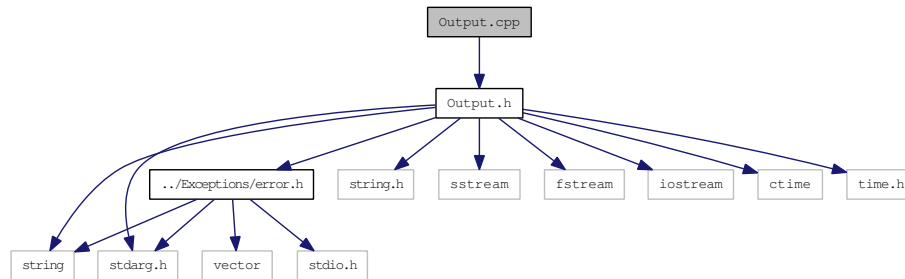


9.30 Output.cpp File Reference

Logging and screen output.

```
#include "Output.h"
```

Include dependency graph for Output.cpp:



Namespaces

- namespace [Output](#)

Functions

- void [Output::open_log_file](#) (string filename)
- void [Output::close_log_file](#) ()
- void [Output::set_output_lvl](#) (int new_outputLvl)
- void [Output::echo](#) (int msglvl, const char *format,...)

Variables

- ofstream * [Output::log_file](#)
- int [Output::outputLvl](#) = 0

9.30.1 Detailed Description

Logging and screen output.

..

Todo

Rewrite type conversion functions as "any to string" and organize them.
Move all headers to the header files and code to cpp files (if it is possible everywhere).

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Output.cpp](#).

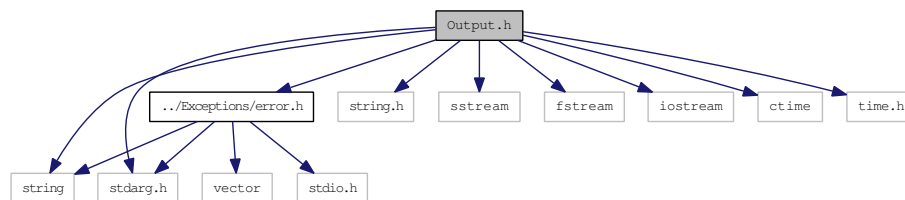
9.31 Output.d File Reference

9.32 Output.h File Reference

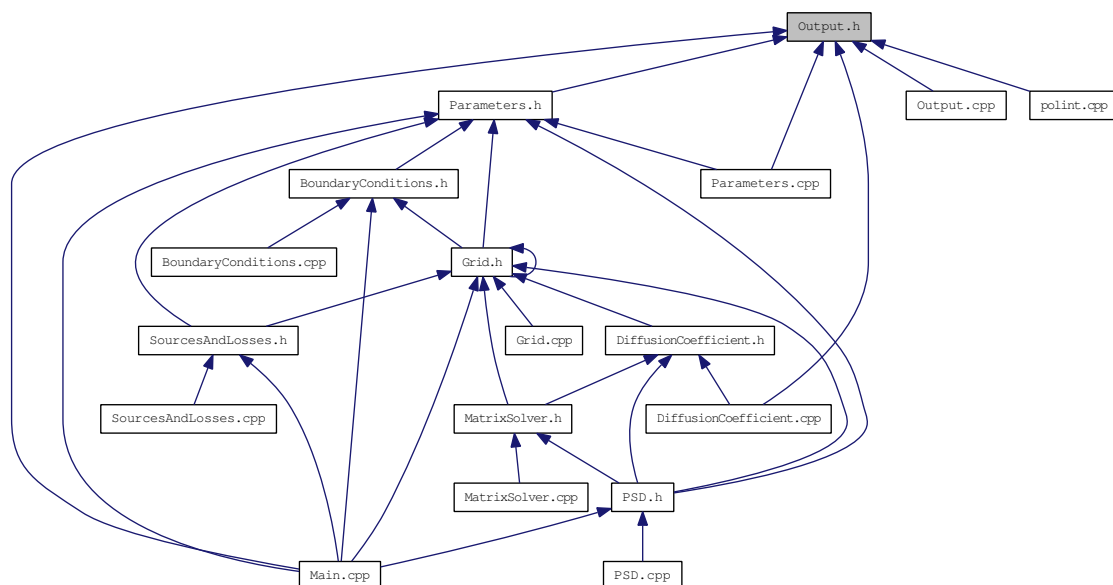
Logging and screen output.

```
#include <string>
#include <string.h>
#include <stdarg.h>
#include <sstream>
#include <fstream>
#include <iostream>
#include <ctime>
#include <time.h>
#include "../Exceptions/error.h"
```

Include dependency graph for Output.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [Output](#)

Functions

- void [Output::open_log_file](#) (string filename)
- void [Output::close_log_file](#) ()
- void [Output::echo](#) (int msglvl, const char *format,...)
- void [Output::set_output_lvl](#) (int new_outputLvl)

9.32.1 Detailed Description

Logging and screen output.

The file provide output functions. [Output](#) usually goes to a screen and a file simultaneously.

[Todo](#)

Rewrite all output with streams insted of old-c functions.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Output.h](#).

9.33.1 Detailed Description

Loads parameters.

Loads parameters from file into map, then use parameters-structure related functions to get parameters values from the map and store them into the parameters structures.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Parameters.cpp](#).

9.33.2 Function Documentation

9.33.2.1 string bool2str (bool *b*)

Converting boolean to string.

Definition at line 522 of file Parameters.cpp.

9.33.2.2 void load_1d (Matrix1D< double > &*var*, string *filename*, double *dt*, int *var_size*)

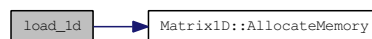
Read 1d matrix_array from txt-file and interpolate to out time-axis.

Definition at line 529 of file Parameters.cpp.

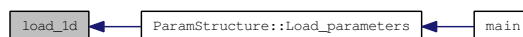
References Matrix1D< T >::AllocateMemory(), i, and Matrix1D< T >::size_x.

Referenced by ParamStructure::Load_parameters().

Here is the call graph for this function:



Here is the caller graph for this function:



9.33.2.3 template<typename T > bool ReadFromFile (T &*variable*, ifstream &*parameters_file*, string *variable_name*, string *default_value* = "") [inline]

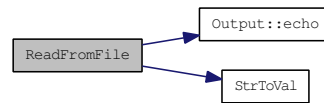
Read each parameter from parameters file.

Definition at line 24 of file Parameters.cpp.

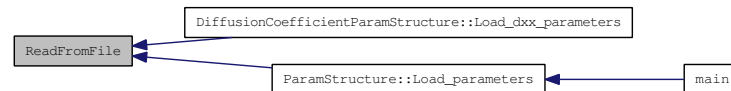
References Output::echo(), and StrToVal().

Referenced by DiffusionCoefficientParamStructure::Load_dxx_parameters(), and ParamStructure::Load_parameters().

Here is the call graph for this function:



Here is the caller graph for this function:



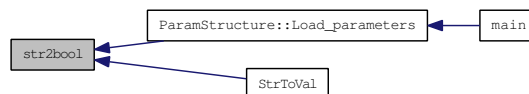
9.33.2.4 `bool str2bool (string str)`

Converting string to boolean.

Definition at line 514 of file `Parameters.cpp`.

Referenced by `ParamStructure::Load_parameters()`, and `StrToVal()`.

Here is the caller graph for this function:



9.33.2.5 `void StrToVal (string input, bool & place)`

Converting string to bool.

Definition at line 505 of file `Parameters.cpp`.

References `str2bool()`.

Here is the call graph for this function:



9.33.2.6 `void StrToVal (string input, string & place)`

Converting string to string.

Function `StrToVal` is used in template, so we need that function to make template function works in case of string.

Definition at line 497 of file `Parameters.cpp`.

9.33.2.7 void StrToVal (string *input*, int & *place*)

Converting string to int.

Definition at line 489 of file Parameters.cpp.

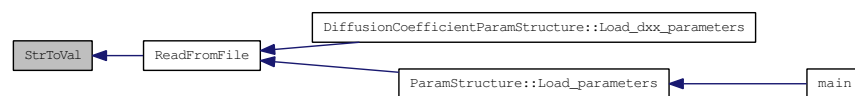
9.33.2.8 void StrToVal (string *input*, double & *place*)

Converting string to double.

Definition at line 481 of file Parameters.cpp.

Referenced by ReadFromFile().

Here is the caller graph for this function:



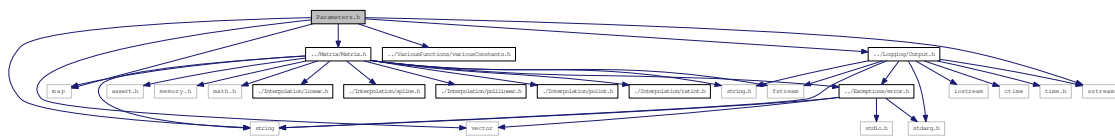
9.34 Parameters.d File Reference

9.35 Parameters.h File Reference

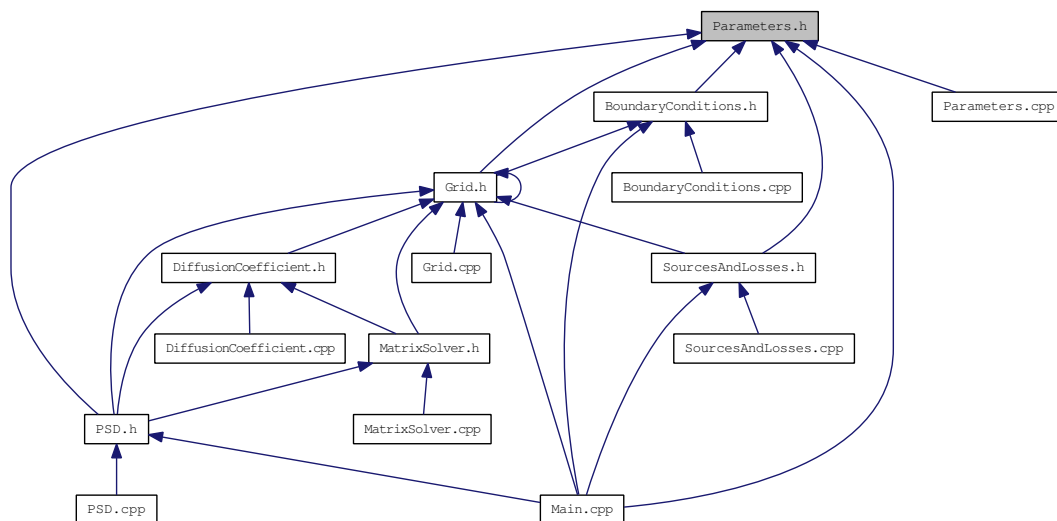
All parameters, loaded from .ini file are described here in one structure.

```
#include <string>
#include <vector>
#include <map>
#include "../Matrix/Matrix.h"
#include "../VariousFunctions/variousConstants.h"
#include "../Logging/Output.h"
#include <sstream>
```

Include dependency graph for Parameters.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct `DiffusionCoefficientParamStructure`
Diffusion coefficient parameters.
- struct `ParamStructure`
Main parameters structure.
- struct `ParamStructure::General_Output_parameters`

General programm output parameters structure.

- struct [ParamStructure::GridElement](#)
Grid element parameters structure.
- struct [ParamStructure::BoundaryCondition](#)
Boundary conditions parameters structure.
- struct [ParamStructure::PSD](#)
PSD parameters structure.
- struct [ParamStructure::SL](#)
Sources and losses.
- struct [ParamStructure::Interpolation](#)
Interpolation parameters structure

Typedefs

- typedef std::vector< [DiffusionCoefficientParamStructure](#) > [DiffusionCoefficientParamStructureList](#)
List of diffusion coefficient parameters.

Functions

- void [StrToVal](#) (string input, double &place)
Converting string to double.
- void [StrToVal](#) (string input, int &place)
Converting string to int.
- void [StrToVal](#) (string input, string &place)
Converting string to string.
- void [StrToVal](#) (string input, bool &place)
Converting string to bool.
- void [load_1d](#) ([Matrix1D](#)< double > &var, string filename, double dt, int var_size=0)
Read 1d matrix_array from txt-file and interpolate to out time-axis.
- bool [str2bool](#) (string str)
Converting string to boolean.
- string [bool2str](#) (bool b)
Converting boolean to string.

9.35.1 Detailed Description

All parameters, loaded from .ini file are described here in one structure.

Each brunch of the structure holds parameters for one object (class) initialization. So, we need to pass one structure (brunch of the main structure) to the classes constructors.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [Parameters.h](#).

9.35.2 Typedef Documentation

9.35.2.1 `typedef std::vector<DiffusionCoefficientParamStructure> DiffusionCoefficientParamStructureList`

List of diffusion coefficient parameters.

Definition at line 79 of file Parameters.h.

9.35.3 Function Documentation

9.35.3.1 `string bool2str (bool b)`

Converting boolean to string.

Definition at line 522 of file Parameters.cpp.

9.35.3.2 `void load_1d (Matrix1D< double > &var, string filename, double dt, int var_size = 0)`

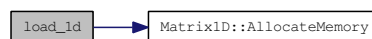
Read 1d matrix_array from txt-file and interpolate to out time-axis.

Definition at line 529 of file Parameters.cpp.

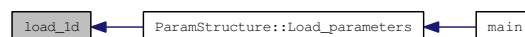
References `Matrix1D< T >::AllocateMemory()`, `i`, and `Matrix1D< T >::size_x`.

Referenced by `ParamStructure::Load_parameters()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.35.3.3 bool str2bool (string *str*)

Converting string to boolean.

Definition at line 514 of file Parameters.cpp.

9.35.3.4 void StrToVal (string *input*, bool & *place*)

Converting string to bool.

Definition at line 505 of file Parameters.cpp.

References str2bool().

Here is the call graph for this function:



9.35.3.5 void StrToVal (string *input*, string & *place*)

Converting string to string.

Function StrToVal is used in template, so we need that function to make template function works in case of string.

Definition at line 497 of file Parameters.cpp.

9.35.3.6 void StrToVal (string *input*, int & *place*)

Converting string to int.

Definition at line 489 of file Parameters.cpp.

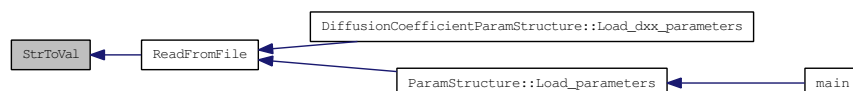
9.35.3.7 void StrToVal (string *input*, double & *place*)

Converting string to double.

Definition at line 481 of file Parameters.cpp.

Referenced by ReadFromFile().

Here is the caller graph for this function:

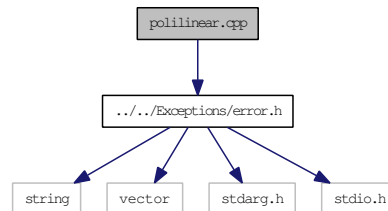


9.36 polilinear.cpp File Reference

Polilinear interpolation sources.

```
#include "../..//Exceptions/error.h"
```

Include dependency graph for polilinear.cpp:



Functions

- double [polilinear](#) (double *xa, double *ya, int n, double x, double lb, double ub)

Do polinomial interpolation between 3 points for all data exepts the boundaries.

9.36.1 Detailed Description

Polilinear interpolation sources.

Author:

Y.Shprits

Definition in file [polilinear.cpp](#).

9.36.2 Function Documentation

9.36.2.1 double polilinear (double *xa, double *ya, int n, double x, double lb, double ub)

Do polinomial interpolation between 3 points for all data exepts the boundaries.

Linear interpolation on the boundaries. Boundary values for extrapolation.

Definition at line 13 of file `polilinear.cpp`.

References [i](#).

Referenced by `Matrix1D< T >::Polilinear()`.

Here is the caller graph for this function:

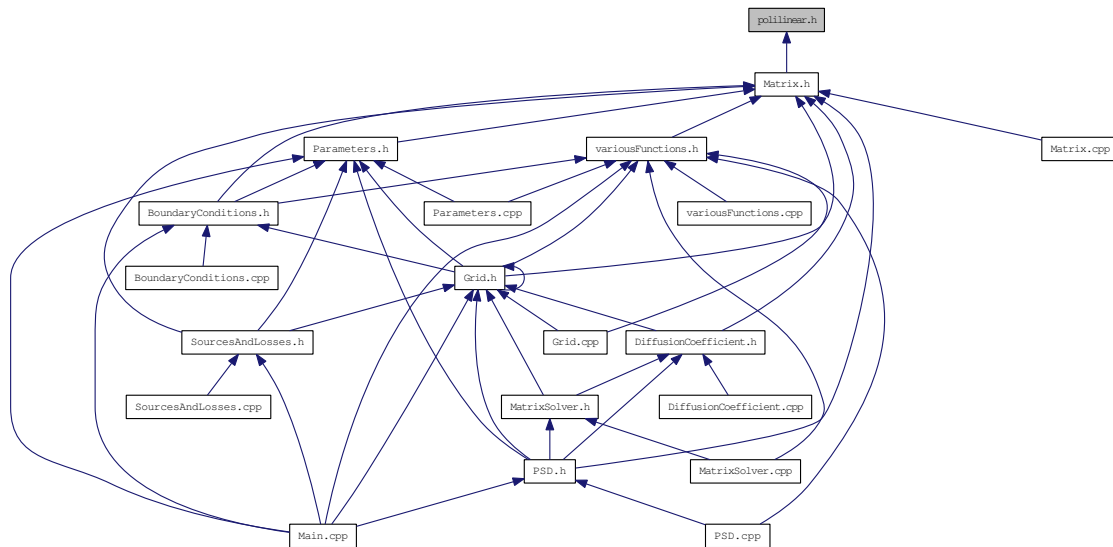


9.37 polilinear.d File Reference

9.38 polilinear.h File Reference

Polilinear interpolation headers.

This graph shows which files directly or indirectly include this file:



Functions

- double [polilinear](#) (double *xa, double *ya, int n, double x, double ub, double lb)

Do polinomial interpolation between 3 points for all data exepts the boundaries.

9.38.1 Detailed Description

Polilinear interpolation headers.

Definition in file [polilinear.h](#).

9.38.2 Function Documentation

9.38.2.1 double [polilinear](#) (double *xa, double *ya, int n, double x, double lb, double ub)

Do polinomial interpolation between 3 points for all data exepts the boundaries.

Linear interpolation on the boundaries. Boundary values for extrapolation.

Definition at line 13 of file polilinear.cpp.

References [i](#).

Referenced by [Matrix1D< T >::Polilinear\(\)](#).

Here is the caller graph for this function:

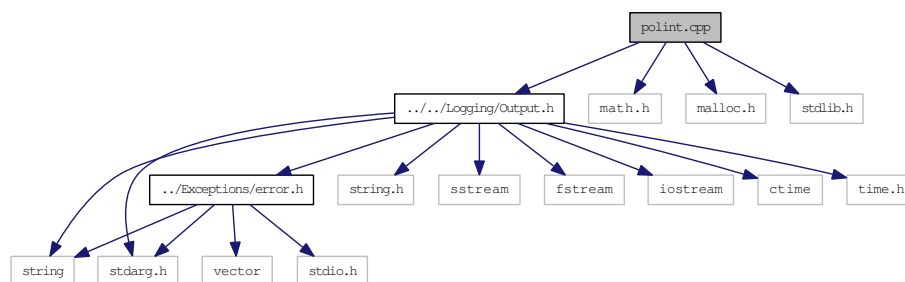


9.39 polint.cpp File Reference

Some other interpolation from numerical recepies.

```
#include "../Logging/Output.h"
#include "math.h"
#include <malloc.h>
#include <stdlib.h>
```

Include dependency graph for polint.cpp:



Functions

- void [polint](#) (double *xa, double *ya, int n, double x, double *y, double *dy)

Some other interpolation from numerical recepies.

9.39.1 Detailed Description

Some other interpolation from numerical recepies.

Definition in file [polint.cpp](#).

9.39.2 Function Documentation

9.39.2.1 void polint (double * xa, double * ya, int n, double x, double * y, double * dy)

Some other interpolation from numerical recepies.

Definition at line 14 of file polint.cpp.

References `Output::echo()`, and `i`.

Referenced by `Matrix1D< T >::Polint()`.

Here is the call graph for this function:



Here is the caller graph for this function:

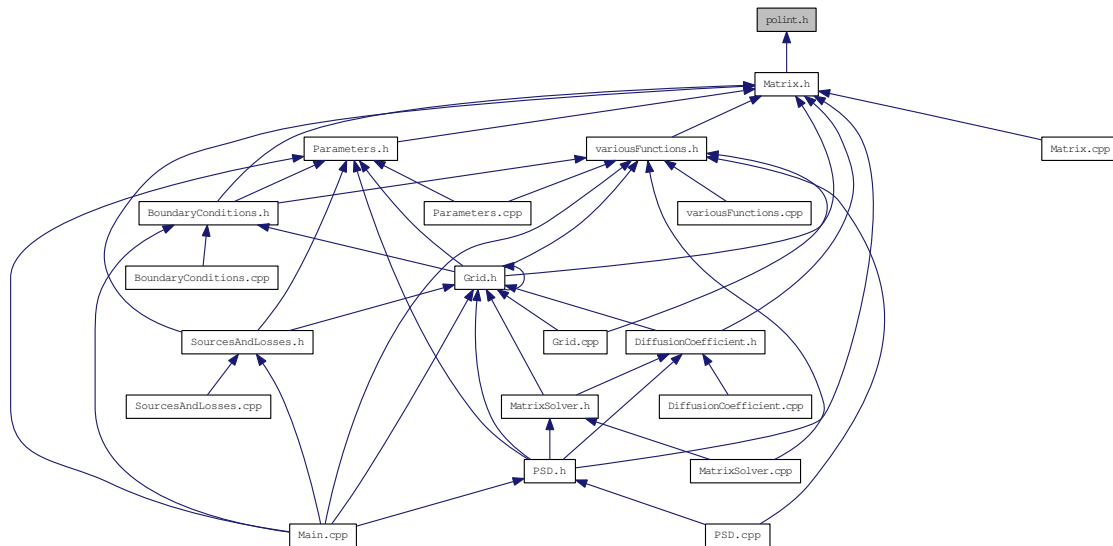


9.40 polint.d File Reference

9.41 polint.h File Reference

Some other interpolation from numerical recepies.

This graph shows which files directly or indirectly include this file:



Functions

- void [polint](#) (double *xa, double *ya, int n, double x, double *y, double *dy)
Some other interpolation from numerical recepies.

9.41.1 Detailed Description

Some other interpolation from numerical recepies.

Definition in file [polint.h](#).

9.41.2 Function Documentation

9.41.2.1 void polint (double * xa, double * ya, int n, double x, double * y, double * dy)

Some other interpolation from numerical recepies.

Definition at line 14 of file polint.cpp.

References `Output::echo()`, and `i`.

Referenced by `Matrix1D< T >::Polint()`.

Here is the call graph for this function:



Here is the caller graph for this function:

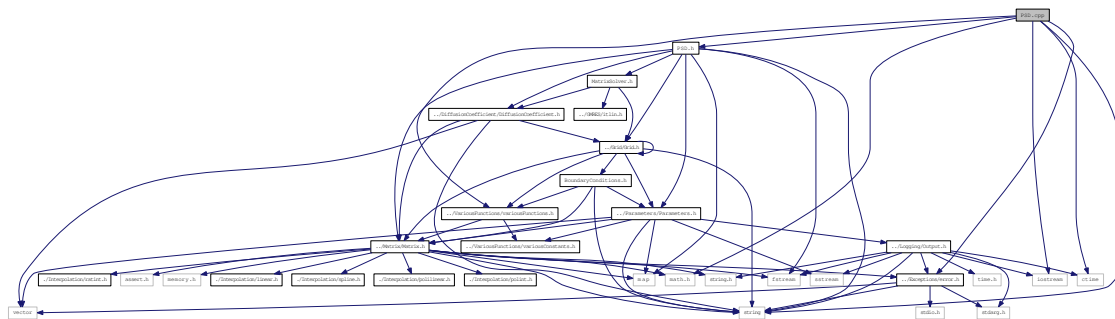


9.42 PSD.cpp File Reference

Makes operations with PSD (like, diffusion).

```
#include "PSD.h"
#include <math.h>
#include "../VariousFunctions/variousFunctions.h"
#include "../Exceptions/error.h"
#include <iostream>
#include <string>
#include <ctime>
```

Include dependency graph for PSD.cpp:



Functions

- void [checkInf](#) ([Matrix3D](#)< double > arr, int il, int im, int ia, double maxNum=1e99)
Checking for infinity value appears during interpolation for 3D arrays.
- void [checkInf](#) ([Matrix1D](#)< double > arr, int il, int im, int ia, double maxNum=1e99)
Checking for infinity value appears during interpolation for 1D arrays.
- void [steady_state](#) ([Matrix1D](#)< double > &f, double tau, double Kp, int nx, [Matrix1D](#)< double > &L, double f_bnd_out, double f_bnd_in)
Calculate steady state.

9.42.1 Detailed Description

Makes operations with PSD (like, diffusion).

Todo

A lot of corrections should be done in [PSD](#) class to make it more logical and less spread.

Author:

Developed under supervision of the PI Yuri Shprits

General view of F-P eq: $Ax = Bx + C A$, B, C referred as MatrixA, MatrixB and MatrixC in the code
 Definition in file [PSD.cpp](#).

9.42.2 Function Documentation

9.42.2.1 void checkInf (Matrix1D< double > *arr*, int *il*, int *im*, int *ia*, double *maxNum* = 1e99)

Checking for infinity value appears during interpolation for 1D arrays.

Parameters:

arr - array of values
il - index
im - index
ia - index
maxNum = 1e99 - max number (to compare with)

Definition at line 982 of file PSD.cpp.

9.42.2.2 void checkInf (Matrix3D< double > *arr*, int *il*, int *im*, int *ia*, double *maxNum* = 1e99)

Checking for infinity value appears during interpolation for 3D arrays.

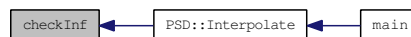
Parameters:

arr - array of values
il - index
im - index
ia - index
maxNum = 1e99 - max number (to compare with)

Definition at line 966 of file PSD.cpp.

Referenced by PSD::Interpolate().

Here is the caller graph for this function:



9.42.2.3 void steady_state (Matrix1D< double > &*f*, double *tau*, double *Kp*, int *nx*, Matrix1D< double > &*L*, double *f_bnd_out*, double *f_bnd_in*)

Calculate steady state.

Compute steady state solution for 1D diffusion.

Parameters:

&*f* - function

tau - life time

Kp - Kp index value

nx - number of points

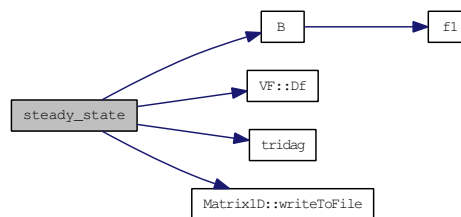
&CL - grid

Definition at line 1510 of file PSD.cpp.

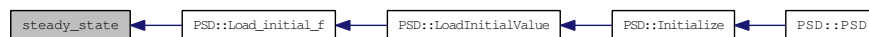
References B(), VF::Df(), i, tridag(), and Matrix1D< T >::writeToFile().

Referenced by PSD::Load_initial_f().

Here is the call graph for this function:



Here is the caller graph for this function:



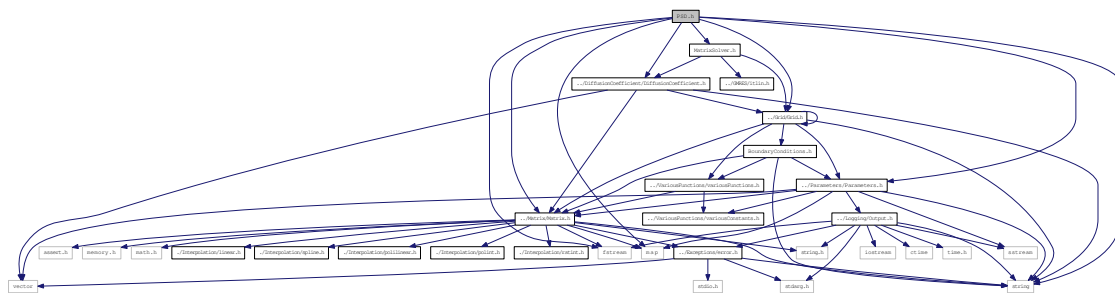
9.43 PSD.d File Reference

9.44 PSD.h File Reference

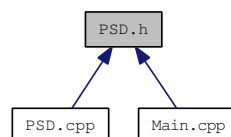
Phase Space Density (PSD).

```
#include <fstream>
#include <string>
#include <map>
#include "../Grid/Grid.h"
#include "../Matrix/Matrix.h"
#include "../DiffusionCoefficient/DiffusionCoefficient.h"
#include "../Parameters/Parameters.h"
#include "MatrixSolver.h"
```

Include dependency graph for PSD.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [PSD](#)
Phase Space Density class.

Functions

- void [steady_state](#) ([Matrix1D](#)< double > &f, double tau, double Kp, int nx, [Matrix1D](#)< double > &CL, double f_bnd_out=1, double f_bnd_in=0)
Compute steady state solution for 1D diffusion.

9.44.1 Detailed Description

Phase Space Density (PSD).

Makes operations with PSD (like, diffusion).

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [PSD.h](#).

9.44.2 Function Documentation

9.44.2.1 `void steady_state (Matrix1D< double > &f, double tau, double Kp, int nx, Matrix1D< double > &L, double f_bnd_out, double f_bnd_in)`

Compute steady state solution for 1D diffusion.

Compute steady state solution for 1D diffusion.

Parameters:

&f - function

tau - life time

Kp - Kp index value

nx - number of points

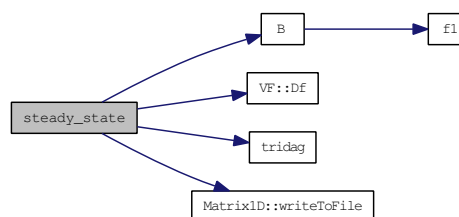
&CL - grid

Definition at line 1510 of file PSD.cpp.

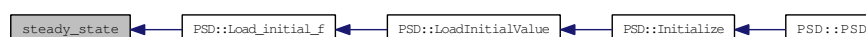
References [B\(\)](#), [VF::Df\(\)](#), [i](#), [tridag\(\)](#), and [Matrix1D< T >::writeToFile\(\)](#).

Referenced by [PSD::Load_initial_f\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

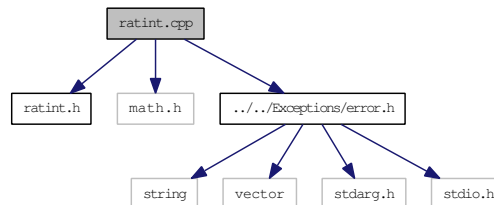


9.45 ratint.cpp File Reference

Some other interpolation.

```
#include "ratint.h"
#include "math.h"
#include "../../Exceptions/error.h"
```

Include dependency graph for ratint.cpp:



Defines

- #define [TINY](#) 1.0e-25

Functions

- void [ratint](#) (double *xa, double *ya, int n, double x, double *y, double *dy)
Some other interpolation.

9.45.1 Detailed Description

Some other interpolation.

Definition in file [ratint.cpp](#).

9.45.2 Define Documentation

9.45.2.1 #define TINY 1.0e-25

Definition at line 11 of file ratint.cpp.

Referenced by ratint().

9.45.3 Function Documentation

9.45.3.1 void ratint (double *xa, double *ya, int n, double x, double *y, double *dy)

Some other interpolation.

Definition at line 16 of file ratint.cpp.

References `i`, and `TINY`.

Referenced by `Matrix1D< T >::Ratint()`.

Here is the caller graph for this function:

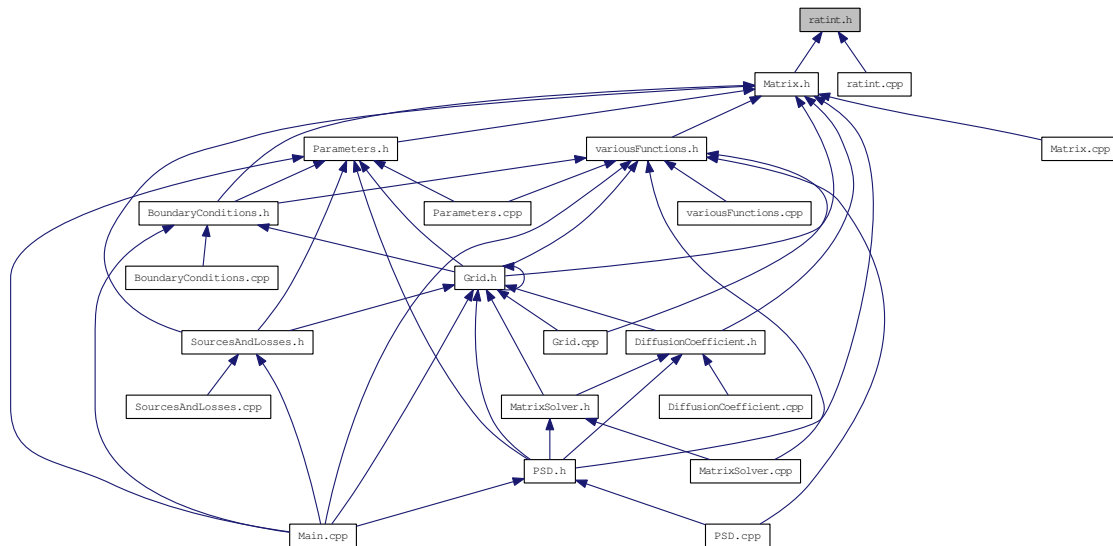


9.46 ratint.d File Reference

9.47 ratint.h File Reference

Some other interpolation.

This graph shows which files directly or indirectly include this file:



Functions

- void [ratint](#) (double *xa, double *ya, int n, double x, double *y, double *dy)
Some other interpolation.

9.47.1 Detailed Description

Some other interpolation.

Definition in file [ratint.h](#).

9.47.2 Function Documentation

9.47.2.1 void ratint (double * xa, double * ya, int n, double x, double * y, double * dy)

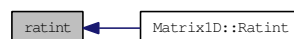
Some other interpolation.

Definition at line 16 of file [ratint.cpp](#).

References [i](#), and [TINY](#).

Referenced by [Matrix1D< T >::Ratint\(\)](#).

Here is the caller graph for this function:

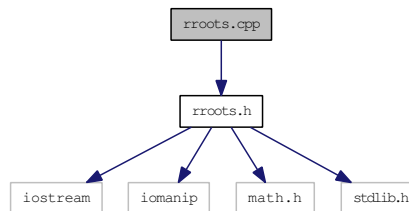


9.48 rroots.cpp File Reference

Finds all roots of polynomial by first finding quadratic factors using Bairstow's method, then extracting roots from quadratics.

```
#include "rroots.h"
```

Include dependency graph for rroots.cpp:



Functions

- `int roots` (double *a, int n, double *wr, double *wi)
Extract individual real or complex roots from list of quadratic factors.
- `void deflate` (double *a, int n, double *b, double *quad, double *err)
Deflate polynomial 'a' by division of 'quad'. Return quotient polynomial in 'b' and error (metric based on remainder) in 'err'.
- `void find_quad` (double *a, int n, double *b, double *quad, double *err, int *iter)
Find quadratic factor using Bairstow's method (quadratic Newton method). A number of ad hoc safeguards are incorporated to prevent stalls due to common difficulties, such as zero slope at iteration point, and convergence problems.
- `void diff_poly` (double *a, int n, double *b)
Differentiate polynomial 'a' returning result in 'b'.
- `void recurse` (double *a, int n, double *b, int m, double *quad, double *err, int *iter)
Attempt to find a reliable estimate of a quadratic factor using modified Bairstow's method with provisions for 'digging out' factors associated with multiple roots.
- `void get_quads` (double *a, int n, double *quad, double *x)
Top level routine to manage the determination of all roots of the given polynomial 'a', returning the quadratic factors (and possibly one linear factor) in 'x'.

9.48.1 Detailed Description

Finds all roots of polynomial by first finding quadratic factors using Bairstow's method, then extracting roots from quadratics.

Implements new algorithm for managing multiple roots.

Date:

(C) 2002, 2003,

Author:

C. Bond. All rights reserved.

Definition in file [rroots.cpp](#).**9.48.2 Function Documentation****9.48.2.1 void deflate (double * *a*, int *n*, double * *b*, double * *quad*, double * *err*)**

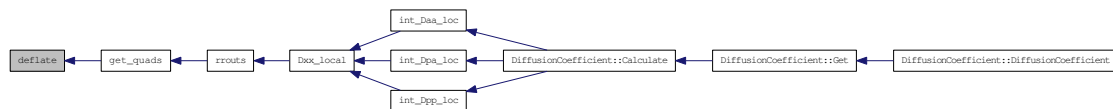
Deflate polynomial '*a*' by division of '*quad*'. Return quotient polynomial in '*b*' and error (metric based on remainder) in '*err*'.

Definition at line 60 of file [rroots.cpp](#).

References [i](#).

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

Here is the caller graph for this function:

**9.48.2.2 void diff_poly (double * *a*, int *n*, double * *b*)**

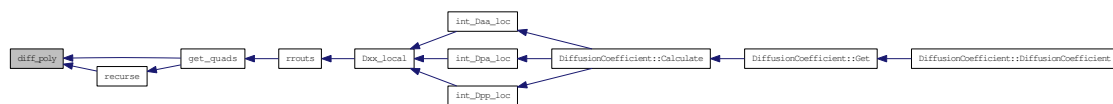
Differentiate polynomial '*a*' returning result in '*b*'.

Definition at line 131 of file [rroots.cpp](#).

References [i](#).

Referenced by [get_quads\(\)](#), and [recurse\(\)](#).

Here is the caller graph for this function:

**9.48.2.3 void find_quad (double * *a*, int *n*, double * *b*, double * *quad*, double * *err*, int * *iter*)**

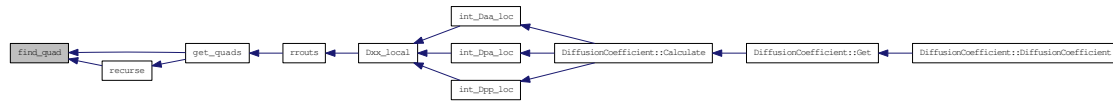
Find quadratic factor using Bairstow's method (quadratic Newton method). A number of ad hoc safeguards are incorporated to prevent stalls due to common difficulties, such as zero slope at iteration point, and convergence problems.

Definition at line 84 of file [rroots.cpp](#).

References `i`, and `maxiter`.

Referenced by `get_quads()`, and `recurse()`.

Here is the caller graph for this function:



9.48.2.4 void get_quads (double * *a*, int *n*, double * *quad*, double * *x*)

Top level routine to manage the determination of all roots of the given polynomial '*a*', returning the quadratic factors (and possibly one linear factor) in '*x*'.

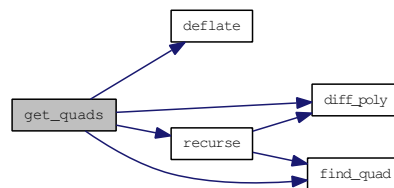
Should gives error in case roots not founded. But that is commented.

Definition at line 201 of file `roots.cpp`.

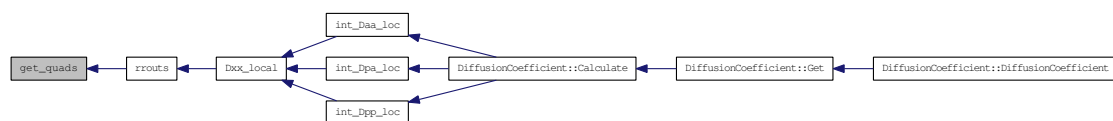
References `deflate()`, `diff_poly()`, `err`, `find_quad()`, `i`, `maxiter`, and `recurse()`.

Referenced by `rroots()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.48.2.5 void recurse (double * *a*, int *n*, double * *b*, int *m*, double * *quad*, double * *err*, int * *iter*)

Attempt to find a reliable estimate of a quadratic factor using modified Bairstow's method with provisions for 'digging out' factors associated with multiple roots.

This resursive routine operates on the principal that differentiation of a polynomial reduces the order of all multiple roots by one, and has no other roots in common with it. If a root of the differentiated polynomial is a root of the original polynomial, there must be multiple roots at that location. The differentiated polynomial, however, has lower order and is easier to solve.

When the original polynomial exhibits convergence problems in the neighborhood of some potential root, a best guess is obtained and tried on the differentiated polynomial. The new best guess is applied recursively

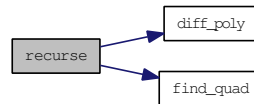
on continually differentiated polynomials until failure occurs. At this point, the previous polynomial is accepted as that with the least number of roots at this location, and its estimate is accepted as the root.

Definition at line 162 of file rroots.cpp.

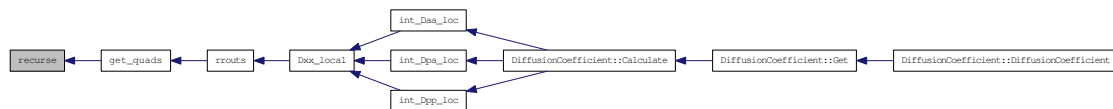
References `diff_poly()`, and `find_quad()`.

Referenced by `get_quads()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.48.2.6 `int roots (double * a, int n, double * wr, double * wi)`

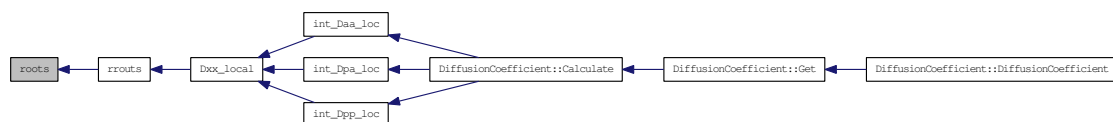
Extract individual real or complex roots from list of quadratic factors.

Definition at line 15 of file rroots.cpp.

References `DBL_EPSILON`.

Referenced by `rroots()`.

Here is the caller graph for this function:



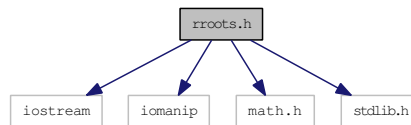
9.49 rroots.d File Reference

9.50 rroots.h File Reference

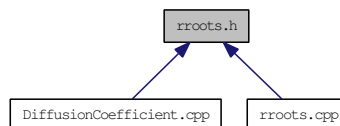
Finds all roots of polynomial by first finding quadratic factors using Bairstow's method, then extracting roots from quadratics.

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include <stdlib.h>
```

Include dependency graph for rroots.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define `maxiter` 5000
maximum number of iterations
- #define `DBL_EPSILON` 1e-15
some other epsilon and stuff

Functions

- int `roots` (double *a, int n, double *wr, double *wi)
Extract individual real or complex roots from list of quadratic factors.
- void `get_quads` (double *a, int n, double *quad, double *x)
Top level routine to manage the determination of all roots of the given polynomial 'a', returning the quadratic factors (and possibly one linear factor) in 'x'.

9.50.1 Detailed Description

Finds all roots of polynomial by first finding quadratic factors using Bairstow's method, then extracting roots from quadratics.

Implements new algorithm for managing multiple roots.

Date:

(C) 2002, 2003,

Author:

C. Bond. All rights reserved.

Definition in file [roots.h](#).

9.50.2 Define Documentation

9.50.2.1 #define DBL_EPSILON 1e-15

some other epsilon and stuff

Definition at line 22 of file roots.h.

Referenced by roots().

9.50.2.2 #define maxiter 5000

maximum number of iterations

Definition at line 20 of file roots.h.

Referenced by find_quad(), and get_quads().

9.50.3 Function Documentation

9.50.3.1 void get_quads (double * *a*, int *n*, double * *quad*, double * *x*)

Top level routine to manage the determination of all roots of the given polynomial '*a*', returning the quadratic factors (and possibly one linear factor) in '*x*'.

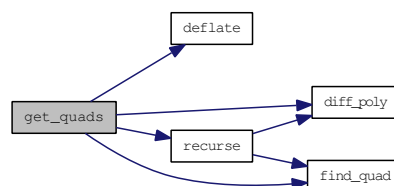
Should gives error in case roots not founded. But that is commented.

Definition at line 201 of file roots.cpp.

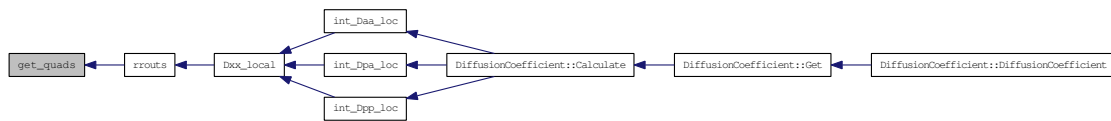
References deflate(), diff_poly(), err, find_quad(), i, maxiter, and recurse().

Referenced by rroots().

Here is the call graph for this function:



Here is the caller graph for this function:



9.50.3.2 int roots (double * *a*, int *n*, double * *wr*, double * *wi*)

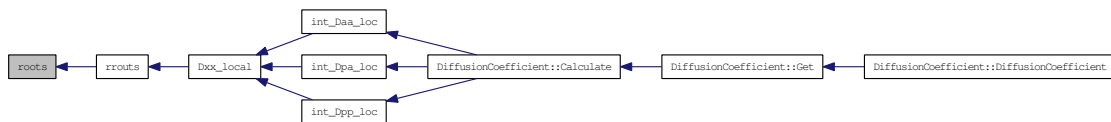
Extract individual real or complex roots from list of quadratic factors.

Definition at line 15 of file roots.cpp.

References DBL_EPSILON.

Referenced by rroots().

Here is the caller graph for this function:



9.52 SourcesAndLosses.d File Reference

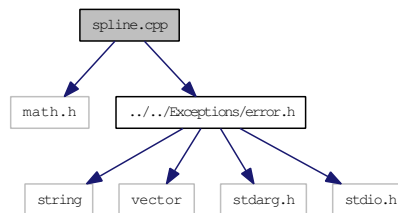
9.54 spline.cpp File Reference

Spline interpolation.

```
#include <math.h>
```

```
#include "../Exceptions/error.h"
```

Include dependency graph for spline.cpp:



Functions

- void [spline](#) (double *x, double *y, int n, double yp1, double ypn, double *y2)
Spline interpolation - calculation of an array with second derivatives (called only once for each interpolation of an array).
- void [splint](#) (double *xa, double *ya, double *y2a, int n, double x, double *y)
Spline interpolation.

9.54.1 Detailed Description

Spline interpolation.

Definition in file [spline.cpp](#).

9.54.2 Function Documentation

9.54.2.1 void spline (double * x, double * y, int n, double yp1, double ypn, double * y2)

Spline interpolation - calculation of an array with second derivatives (called only once for each interpolation of an array).

Given arrays $x[1..n]$ and $y[1..n]$ containing a tabulated function, i.e., $y_i = f(x_i)$, with $x_1 < x_2 < \dots < x_N$, and given values $yp1$ and ypn for the first derivative of the interpolating function at points 1 and n , respectively, this routine returns an array $y2[1..n]$ that contains the second derivatives of the interpolating function at the tabulated points x_i . If $yp1$ and/or ypn are equal to $1 - 10^{30}$ or larger, the routine is signaled to set the corresponding boundary condition for a natural spline, with zero second derivative on that boundary.

Parameters:

double *x - old x

double *y - old function

int n - number of points
double yp1 - derivative on boundary / flag if $> 10^{30}$
double ypn - derivative on boundary / flag if $> 10^{30}$
double *y2 - array with second derivatives of the function

Definition at line 30 of file spline.cpp.

References i.

Referenced by Matrix1D< T >::Spline().

Here is the caller graph for this function:



9.54.2.2 void splint (double * xa, double * ya, double * y2a, int n, double x, double * y)

Spline interpolation.

Given the arrays xa[1..n] and ya[1..n], which tabulate a function (with the xai -s in order), and given the array y2a[1..n], which is the output from spline above, and given a value of x, this routine returns a cubic-spline interpolated value y.

Parameters:

double *xa - old grid array
double *ya - old function array
double *y2a - old function second derivatives array
int n - size of arrays
double x - new grid point
double *y - new grid value

Definition at line 86 of file spline.cpp.

References nrerror().

Referenced by Matrix1D< T >::Spline().

Here is the call graph for this function:



Here is the caller graph for this function:

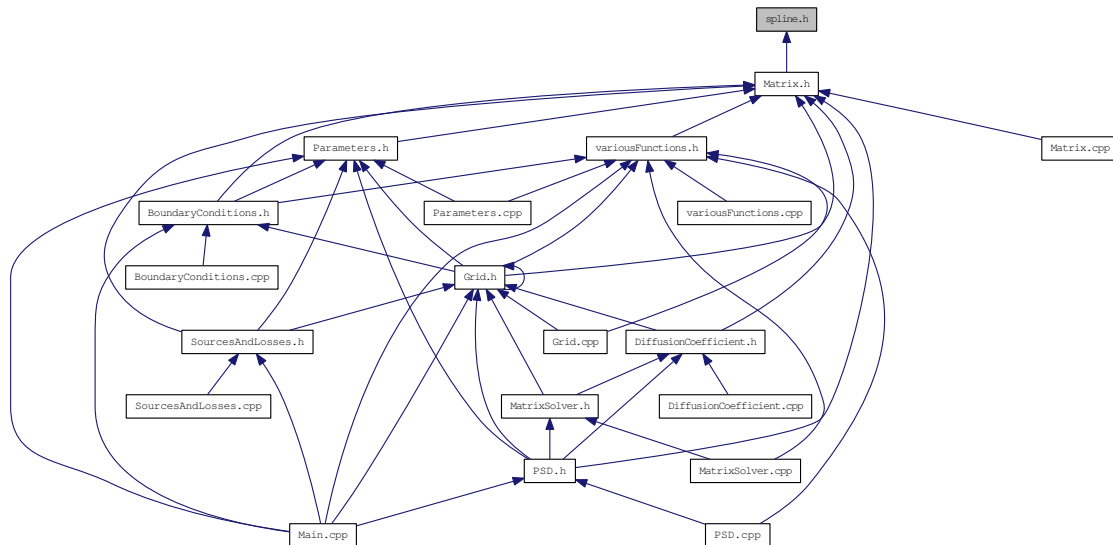


9.55 spline.d File Reference

9.56 spline.h File Reference

Spline interpolation.

This graph shows which files directly or indirectly include this file:



Functions

- void [spline](#) (double *x, double *y, int n, double yp1, double ypn, double *y2)
Spline interpolation - calculation of an array with second derivatives (called only once for each interpolation of an array).
- void [splint](#) (double *xa, double *ya, double *y2a, int n, double x, double *y)
Spline interpolation.

9.56.1 Detailed Description

Spline interpolation.

Definition in file [spline.h](#).

9.56.2 Function Documentation

9.56.2.1 void spline (double * x, double * y, int n, double yp1, double ypn, double * y2)

Spline interpolation - calculation of an array with second derivatives (called only once for each interpolation of an array).

Given arrays $x[1..n]$ and $y[1..n]$ containing a tabulated function, i.e., $y_i = f(x_i)$, with $x_1 < x_2 < \dots < x_N$, and given values $yp1$ and ypn for the first derivative of the interpolating function at points 1 and n , respectively, this routine returns an array $y2[1..n]$ that contains the second derivatives of the interpolating

function at the tabulated points x_i . If $yp1$ and/or ypn are equal to $1 - 10^{30}$ or larger, the routine is signaled to set the corresponding boundary condition for a natural spline, with zero second derivative on that boundary.

Parameters:

double *x - old x
double *y - old function
int n - number of points
double yp1 - derivative on boundary / flag if $> 10^{30}$
double ypn - derivative on boundary / flag if $> 10^{30}$
double *y2 - array with second derivatives of the function

Definition at line 30 of file spline.cpp.

References i.

Referenced by Matrix1D< T >::Spline().

Here is the caller graph for this function:



9.56.2.2 void splint (double *xa, double *ya, double *y2a, int n, double x, double *y)

Spline interpolation.

Given the arrays $xa[1..n]$ and $ya[1..n]$, which tabulate a function (with the x_{ai} -s in order), and given the array $y2a[1..n]$, which is the output from spline above, and given a value of x , this routine returns a cubic-spline interpolated value y .

Parameters:

double *xa - old grid array
double *ya - old function array
double *y2a - old function second derivatives array
int n - size of arrays
double x - new grid point
double *y - new grid value

Definition at line 86 of file spline.cpp.

References nerror().

Referenced by Matrix1D< T >::Spline().

Here is the call graph for this function:



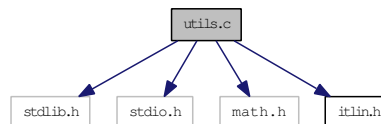
Here is the caller graph for this function:



9.57 utils.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "itlin.h"
```

Include dependency graph for utils.c:



Defines

- `#define TOLMIN 1.0e-15`
- `#define TOLMAX 1.0e-1`

Functions

- `int zibnum_fwalloc` (int size, double **ptr, char vname[])
- `int zibnum_iwalloc` (int size, int **ptr, char vname[])
- `int zibnum_pfwalloc` (int size, double ***ptr, char vname[])
- `double zibnum_scaled_norm2` (int n, double *v, double *scale)
- `double zibnum_scaled_sprod` (int n, double *v1, double *v2, double *scale)
- `double zibnum_norm2` (int n, double *v)
- `void zibnum_scale` (int n, double *v1, double *v2, double *scale)
- `void zibnum_descale` (int n, double *v1, double *v2, double *scale)
- `void itlin_noprecon` (int n, double *x, double *z)
- `void itlin_dataout` (int k, int n, double *x, struct [ITLIN_DATA](#) *data)
- `int itlin_parcheck_and_print` (int n, [MATVEC](#) *matvec, struct [ITLIN_OPT](#) *opt, int itlin_code)

Variables

- struct [ITLIN_IO](#) * `itlin_ioctl` = NULL

9.57.1 Define Documentation

9.57.1.1 `#define TOLMAX 1.0e-1`

Referenced by `itlin_parcheck_and_print()`.

9.57.1.2 `#define TOLMIN 1.0e-15`

Referenced by `itlin_parcheck_and_print()`.

9.57.2 Function Documentation

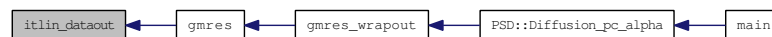
9.57.2.1 void itlin_dataout (int *k*, int *n*, double * *x*, struct ITLIN_DATA * *data*)

Definition at line 286 of file utils.c.

References ITLIN_DATA::codeid, DATA, DATALEVEL, FITER, FMISC, FRES, i, ITLIN_DATA::mode, ITLIN_DATA::normdx, ITLIN_DATA::res, ITLIN_DATA::residuum, ITLIN_DATA::t, and ITLIN_DATA::tau.

Referenced by gmres().

Here is the caller graph for this function:

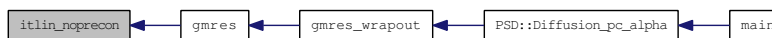


9.57.2.2 void itlin_noprecon (int *n*, double * *x*, double * *z*)

Definition at line 281 of file utils.c.

Referenced by gmres().

Here is the caller graph for this function:



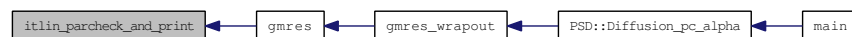
9.57.2.3 int itlin_parcheck_and_print (int *n*, MATVEC * *matvec*, struct ITLIN_OPT * *opt*, int *itlin_code*)

Definition at line 329 of file utils.c.

References Absolute, ITLIN_OPT::convcheck, ERROR, ERRORLEVEL, i, ITLIN_OPT::i_max, ITLIN_OPT::maxiter, MIN, MONITOR, MONITORLEVEL, Relative, ITLIN_OPT::rho, ITLIN_OPT::scale, SMALL, ITLIN_OPT::tol, TOLMAX, and TOLMIN.

Referenced by gmres().

Here is the caller graph for this function:



9.57.2.4 void zibnum_descale (int *n*, double * *v1*, double * *v2*, double * *scale*)

Definition at line 275 of file utils.c.

References i.

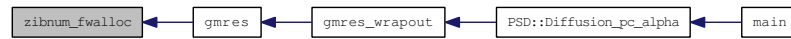
9.57.2.5 int zibnum_fwalloc (int *size*, double *ptr*, char *vname*[])**

Definition at line 196 of file utils.c.

References `ERROR`, `ERRORLEVEL`, and `i`.

Referenced by `gmres()`.

Here is the caller graph for this function:

**9.57.2.6 int zibnum_iwalloc (int *size*, int ***ptr*, char *vname*[])**

Definition at line 213 of file utils.c.

References `ERROR`, `ERRORLEVEL`, and `i`.

9.57.2.7 double zibnum_norm2 (int *n*, double **v*)

Definition at line 262 of file utils.c.

References `i`.

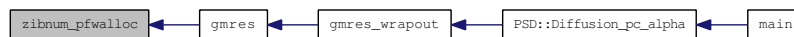
9.57.2.8 int zibnum_pfwalloc (int *size*, double **ptr*, char *vname*[])**

Definition at line 230 of file utils.c.

References `ERROR`, `ERRORLEVEL`, and `i`.

Referenced by `gmres()`.

Here is the caller graph for this function:

**9.57.2.9 void zibnum_scale (int *n*, double **v1*, double **v2*, double **scale*)**

Definition at line 269 of file utils.c.

References `i`.

9.57.2.10 double zibnum_scaled_norm2 (int *n*, double **v*, double **scale*)

Definition at line 247 of file utils.c.

References `i`.

9.57.2.11 double zibnum_scaled_sprod (int *n*, double * *v1*, double * *v2*, double * *scale*)

Definition at line 254 of file utils.c.

References [i](#).

9.57.3 Variable Documentation**9.57.3.1 struct ITLIN_IO* itlin_ioctl = NULL**

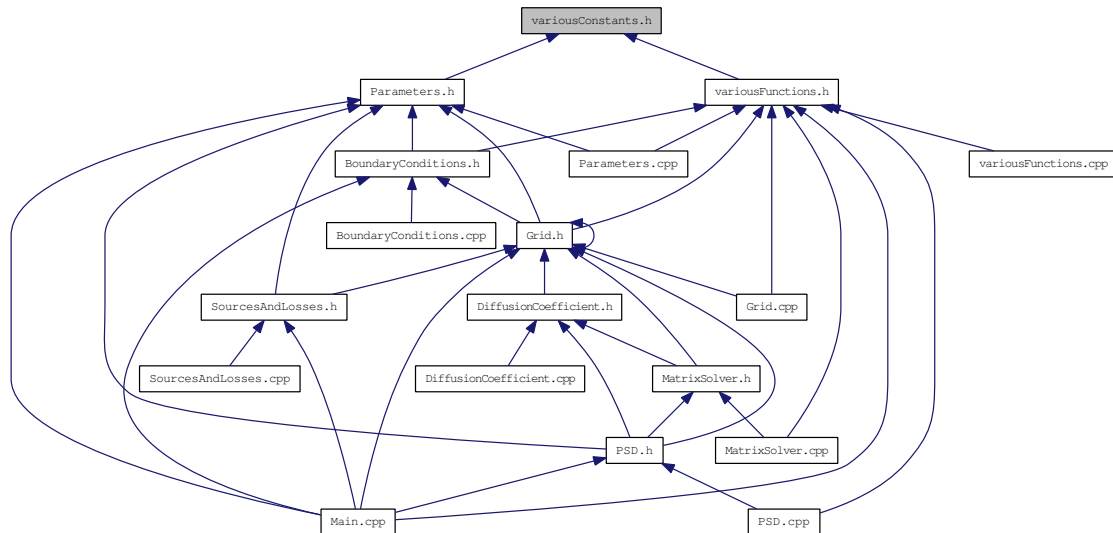
Definition at line 194 of file utils.c.

9.58 utils.d File Reference

9.59 variousConstants.h File Reference

Various usefull constants.

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [VC](#)
Various constants.

9.59.1 Detailed Description

Various usefull constants.

Author:

Developed under supervision of the PI Yuri Shprits

Definition in file [variousConstants.h](#).

- double [VF::J_L7_corrected](#) (double K)
Specifying outer boundary, more accurate function.
- double [VF::mu2pc](#) (double L, double mu, double alpha)
Calculating pc from L, mu, alpha.
- double [VF::pc2mu](#) (double L, double pc, double alpha)
Calculating mu from L, pc, alpha.
- double [VF::Alpha2J](#) (double L, double pc, double Alpha)
Calculating J from L, pc, Alpha.
- double [VF::alc](#) (double L)
Lost cone in rad.
- double [VF::f_interp](#) (double E, double f1, double E1, double f2, double E2)
Getting f on specific energy.
- double [VF::mu_calc](#) (double L, double pc, double Alpha)
Calculating mu from L, pc, Alpha.
- double [VF::Jc_calc](#) (double L, double pc, double Alpha)
Calculating J from L, pc, Alpha.
- double [VF::Y_old](#) (double alpha)
Y(alpha) function.
- double [VF::Y](#) (double alpha)
Y(alpha) function.
- double [VF::density](#) (double L)
Chorus density model?
- double [VF::max](#) (double v1, double v2)
Chorus density model?
- string [VF::dtostr](#) (double n)
Double to string.

9.60.1 Detailed Description

Various functions.

Author:

Developed by Yuri Shprits

Definition in file [variousFunctions.cpp](#).

9.61 variousFunctions.d File Reference

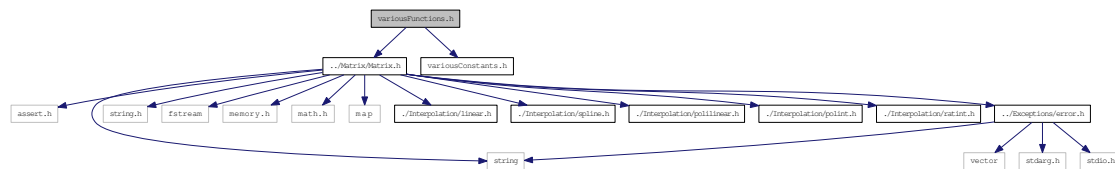
9.62 variousFunctions.h File Reference

Variousse functions.

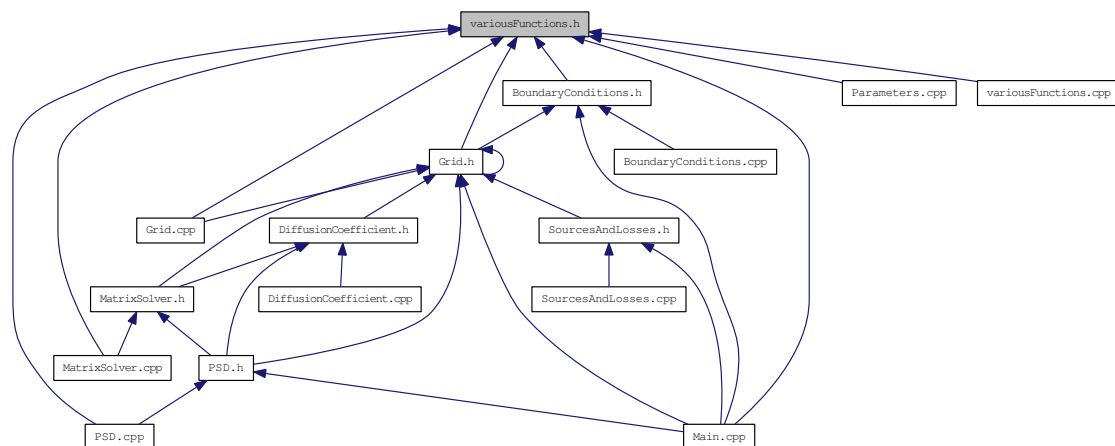
```
#include "../Matrix/Matrix.h"
```

```
#include "variousConstants.h"
```

Include dependency graph for variousFunctions.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **VF**
Various functions.

Functions

- double **VF::G** (double x)
G-function ;-).
- double **VF::B** (double Lparam)
Computation of dipole magnetic field.
- double **VF::Df** (double L, double Kp)
Radial Diffusion coeificeint computed following [Brautigam and Albet , 2000].

- double [VF::pfunc](#) (double K)
Computation of momentum from Kinetic energy.
- double [VF::Kfunc](#) (double pc)
Computation of Kinetic energy from given momentum.
- double [VF::bounce_time](#) (double E, double L)
Bounce time.
- double [VF::J_L7_corrected](#) (double K)
Specifying outer boundary, more accurate function.
- double [VF::J_L7](#) (double K, double x1, double y1, double x2, double y2)
Specifying outer boundary.
- double [VF::mu2pc](#) (double L, double mu, double alpha)
Calculating pc from L, mu, alpha.
- double [VF::pc2mu](#) (double L, double pc, double alpha)
Calculating mu from L, pc, alpha.
- double [VF::alc](#) (double L)
Lost cone in rad.
- double [VF::f_interp](#) (double E, double f1, double E1, double f2, double E2)
Getting f on specific energy.
- double [VF::Jc2Alpha](#) (double L, double pc, double Jc)
- double [VF::Alpha2Jc](#) (double L, double pc, double Alpha)
- double [VF::Y](#) (double alpha)
Y(alpha) function.
- double [VF::mu_calc](#) (double L, double pc, double Alpha)
Calculating mu from L, pc, Alpha.
- double [VF::Jc_calc](#) (double L, double pc, double Alpha)
Calculating J from L, pc, Alpha.
- double [VF::max](#) (double v1, double v2)
Chorus density model?
- string [VF::dtostr](#) (double n)
Double to string.

9.62.1 Detailed Description

Variousse functions.

Author:

Developed by Yuri Shprits

Definition in file [variousFunctions.h](#).

Index

- ~Linear
 - Maths::Interpolation::Linear, [102](#)
- ~Matrix1D
 - Matrix1D, [107](#)
- ~Matrix2D
 - Matrix2D, [117](#)
- ~Matrix3D
 - Matrix3D, [126](#)
- ~PSD
 - PSD, [151](#)
- _CRT_SECURE_NO_DEPRECATED
 - Main.cpp, [232](#)
- abs
 - Matrix3D, [127](#)
- Absolute
 - itlin.h, [226](#)
- ActivateAndScale
 - DiffusionCoefficientsGroup, [67](#)
- add
 - error_msg, [71](#)
- AddBoundary
 - MatrixSolver.cpp, [247](#)
- alc
 - VF, [22](#)
- all
 - Make_boundary.m, [237](#)
- AllocateMemory
 - DiffusionCoefficient, [47](#)
 - GridElement, [84](#)
 - Matrix1D, [107](#)
 - Matrix2D, [117](#)
 - Matrix3D, [127](#)
- Alpha
 - DiffusionCoefficientParamStructure, [58](#)
- alpha
 - Grid, [79](#)
- Alpha2J
 - VF, [22](#)
- Alpha2Jc
 - VF, [23](#)
- alpha_LowerBoundaryCondition
 - ParamStructure, [140](#)
- Alpha_ne
 - DiffusionCoefficient.cpp, [184](#)
 - DiffusionCoefficient.h, [194](#)
- alpha_UpperBoundaryCondition
 - ParamStructure, [140](#)
- alphaSize
 - Grid, [79](#)
- approximationMethod
 - ParamStructure::PSD, [165](#)
- B
 - DiffusionCoefficient.cpp, [185](#)
 - VF, [23](#)
- B_unnorm
 - Make_boundary.m, [237](#)
- BC_parameters
 - BoundaryCondition, [36](#)
- Bf
 - Make_boundary.m, [237](#)
 - ParamStructure, [140](#)
- Bf1
 - Make_boundary.m, [237](#)
- Bf2
 - Make_boundary.m, [237](#)
- Bf3
 - Make_boundary.m, [237](#)
- bisection
 - bisection.cpp, [176](#)
 - bisection.h, [178](#)
- bisection.cpp, [175](#)
 - bisection, [176](#)
- bisection.d, [177](#)
- bisection.h, [178](#)
 - bisection, [178](#)
 - max_error, [178](#)
- bool2str
 - erf.cpp, [201](#)
 - Parameters.cpp, [271](#)
 - Parameters.h, [277](#)
- bounce_time
 - VF, [23](#)
- BoundaryCondition, [31](#)
 - BC_parameters, [36](#)
 - BoundaryCondition, [33](#)
 - calculationType, [36](#)
 - filename, [36](#)
 - Initialize, [33](#)

- initialType, 36
- LoadBoundaryCondition, 33
- MakeBoundaryCondition, 34
- operator*, 34
- operator=, 35
- Update, 35
- value, 36
- BoundaryConditions.cpp, 180
- BoundaryConditions.d, 181
- BoundaryConditions.h, 182
- Bw
 - DiffusionCoefficientParamStructure, 58
- BwFromLambda
 - DiffusionCoefficientParamStructure, 58
- Calculate
 - DiffusionCoefficient, 47
- CalculationMatrix, 39
 - CalculationMatrix, 39
 - change_ind, 41
 - index1d, 40
 - Initialize, 40
 - initialized, 41
 - total_size, 41
 - writeToFile, 41
 - x_size, 41
 - y_size, 41
- calculationType
 - BoundaryCondition, 36
- change_ind
 - CalculationMatrix, 41
 - Matrix3D, 133
- check_time
 - Main.cpp, 232
- CheckEachIter
 - itlin.h, 227
- checkInf
 - PSD.cpp, 289
- CheckOnRestart
 - itlin.h, 227
- close_log_file
 - Output, 17
- code
 - single_error, 168
- codeid
 - ITLIN_DATA, 94
- constBf
 - ParamStructure, 140
- constKp
 - ParamStructure, 140
- constLpp
 - ParamStructure, 140
- CONV_CHECK
 - itlin.h, 226
- convcheck
 - ITLIN_OPT, 99
- d_omega
 - DiffusionCoefficientParamStructure, 58
- Daa_root
 - DiffusionCoefficient.cpp, 185
 - DiffusionCoefficient.h, 195
- dat
 - Make_boundary.m, 238
- DATA
 - itlin.h, 224
- datafile
 - ITLIN_OPT, 99
- DATALEVEL
 - itlin.h, 224
- datalevel
 - ITLIN_OPT, 99
- datfile
 - ITLIN_IO, 97
- datlevel
 - ITLIN_IO, 97
- daxpy_
 - itlin.h, 227
- DBL_EPSILON
 - rroots.h, 304
- Debug
 - itlin.h, 226
- deflate
 - rroots.cpp, 299
- density
 - VF, 24
- Df
 - VF, 24
- DiagMatrix
 - Matrix.h, 244
- diff_poly
 - rroots.cpp, 299
- Diffusion_alpha
 - PSD, 151
- Diffusion_L
 - PSD, 152
- Diffusion_pc
 - PSD, 153
- Diffusion_pc_alpha
 - PSD, 154
- DiffusionCoefficient, 43
 - AllocateMemory, 47
 - Calculate, 47
 - DiffusionCoefficient, 45, 46
 - Dxx_parameters, 55
 - Get, 48
 - is_active, 55
 - LoadDiffusionCoefficient, 49

- LoadDiffusionCoefficientFromFileWithGrid, 50
- LoadDiffusionCoefficientFromPlaneFile, 50
- MakeDLL, 51
- MakeDLL100, 51
- MakeDLL_B, 52
- MakeDLL_BE, 52
- MakeDLL_BE_res, 52
- MakeDLL_FAKE, 53
- operator*, 53
- operator+, 54
- operator=, 54
- Scale, 54
- type, 55
- useScale, 55
- DiffusionCoefficient.cpp, 183
 - Alpha_ne, 184
 - B, 185
 - Daa_root, 185
 - double_zero, 184
 - Dpa_root, 186
 - Dpp_root, 186
 - Dxx_ba, 187
 - Dxx_local, 187
 - f1, 188
 - F_cap, 188
 - F_cap2, 188
 - func_tmp, 189
 - int_Daa_loc, 189
 - int_Dpa_loc, 189
 - int_Dpp_loc, 190
 - min_Dxx, 184
 - quad1, 191
 - rrouts, 191
- DiffusionCoefficient.d, 192
- DiffusionCoefficient.h, 193
 - Alpha_ne, 194
 - Daa_root, 195
 - Dpa_root, 195
 - Dpp_root, 195
 - Dxx_ba, 196
 - Dxx_local, 196
 - f1, 197
 - F_cap, 197
 - func_tmp, 197
 - int_Daa_loc, 197
 - int_Dpa_loc, 198
 - int_Dpp_loc, 198
 - rrouts, 199
- DiffusionCoefficientParamStructure, 56
 - Alpha, 58
 - Bw, 58
 - BwFromLambda, 58
 - d_omega, 58
 - DxxKp, 59
 - DxxName, 59
 - DxxType, 59
 - EMeV, 59
 - eta1, 59
 - eta2, 59
 - eta3, 59
 - f, 60
 - filename, 60
 - filetype, 60
 - L, 60
 - lam_max, 60
 - lam_min, 60
 - Load_dxx_parameters, 58
 - loadOrCalculate, 60
 - MLT_averaging, 60
 - multiplicator, 61
 - nint, 61
 - nu, 61
 - numberDensity, 61
 - omega_lc, 61
 - Omega_m, 61
 - Omega_mType, 61
 - omega_uc, 62
 - particle, 62
 - s, 62
 - time_end, 62
 - time_start, 62
 - useScale, 62
 - waveName, 62
 - waveType, 63
- DiffusionCoefficientParamStructureList
 - Parameters.h, 277
- DiffusionCoefficientsGroup, 64
 - ActivateAndScale, 67
 - DiffusionCoefficientsGroup, 66
 - DxxList, 69
 - Get, 67, 68
 - operator=, 68, 69
- DiffusionMixTermExplicit
 - PSD, 156
- DLLType
 - ParamStructure, 140
- double_zero
 - DiffusionCoefficient.cpp, 184
- Dpa_root
 - DiffusionCoefficient.cpp, 186
 - DiffusionCoefficient.h, 195
- Dpp_root
 - DiffusionCoefficient.cpp, 186
 - DiffusionCoefficient.h, 195
- dtostr
 - VF, 24
- Dxx_ba

- DiffusionCoefficient.cpp, 187
- DiffusionCoefficient.h, 196
- Dxx_local
 - DiffusionCoefficient.cpp, 187
 - DiffusionCoefficient.h, 196
- Dxx_parameters
 - DiffusionCoefficient, 55
- DxxKp
 - DiffusionCoefficientParamStructure, 59
- DxxList
 - DiffusionCoefficientsGroup, 69
- DxxName
 - DiffusionCoefficientParamStructure, 59
- DxxParamStructureList
 - ParamStructure, 141
- DxxType
 - DiffusionCoefficientParamStructure, 59
- echo
 - Output, 17
- EE
 - MatrixSolver.cpp, 247
- EMeV
 - DiffusionCoefficientParamStructure, 59
- epc
 - Grid, 79
- epcSize
 - Grid, 80
- EPMACH
 - itlin.h, 224
- EPS
 - erf.h, 207
- erf
 - erf.cpp, 201
 - erf.h, 207
- erf.cpp, 200
 - bool2str, 201
 - erf, 201
 - gammln, 201
 - gammq, 201
 - gammq, 202
 - gcf, 202
 - gser, 203
 - nerror, 203
 - str2bool, 204
- erf.d, 205
- erf.h, 206
 - EPS, 207
 - erf, 207
 - FPMIN, 207
 - gammln, 208
 - gammq, 208
 - gammq, 208
 - gcf, 209
 - gser, 209
 - ITMAX, 207
- err
 - SourcesAndLosses.cpp, 306
- errfile
 - ITLIN_IO, 97
- errlevel
 - ITLIN_IO, 97
- ERROR
 - itlin.h, 224
- error.h, 211
- error_msg, 70
 - add, 71
 - error_msg, 70, 71
 - error_msg, 70, 71
 - errors_stack, 72
 - what, 72
- errorfile
 - ITLIN_OPT, 99
- ERRORLEVEL
 - itlin.h, 224
- errorlevel
 - ITLIN_OPT, 100
- errors_stack
 - error_msg, 72
- eta1
 - DiffusionCoefficientParamStructure, 59
- eta2
 - DiffusionCoefficientParamStructure, 59
- eta3
 - DiffusionCoefficientParamStructure, 59
- f
 - DiffusionCoefficientParamStructure, 60
- f1
 - DiffusionCoefficient.cpp, 188
 - DiffusionCoefficient.h, 197
- F_cap
 - DiffusionCoefficient.cpp, 188
 - DiffusionCoefficient.h, 197
- F_cap2
 - DiffusionCoefficient.cpp, 188
- f_interp
 - VF, 24
- False
 - itlin.h, 226
- fileBf
 - ParamStructure, 141
- fileKp
 - ParamStructure, 141
- fileLpp
 - ParamStructure, 141
- filename
 - BoundaryCondition, 36

- DiffusionCoefficientParamStructure, 60
- ParamStructure::BoundaryCondition, 38
- fileName1D
 - ParamStructure::General_Output_parameters, 73
- filetype
 - DiffusionCoefficientParamStructure, 60
- Final
 - ITLIN_DATA, 93
- find_alpha
 - Grid.cpp, 218
 - Grid.h, 222
- find_alpha_res
 - Grid.cpp, 218
- find_quad
 - rroots.cpp, 299
- FITER
 - itlin.h, 225
- FMISC
 - itlin.h, 225
- folderName
 - ParamStructure::General_Output_parameters, 73
- for_norm_A
 - MatrixSolver.cpp, 248
- FPMIN
 - erf.h, 207
- free_matrix
 - Matrix.cpp, 240
- FRES
 - itlin.h, 225
- func_tmp
 - DiffusionCoefficient.cpp, 189
 - DiffusionCoefficient.h, 197
- G
 - VF, 25
- gammln
 - erf.cpp, 201
 - erf.h, 208
- gammq
 - erf.cpp, 201
 - erf.h, 208
- gammq
 - erf.cpp, 202
 - erf.h, 208
- gauss_solve
 - MatrixSolver.cpp, 248
 - MatrixSolver.h, 260
- GBIT
 - ITLIN_DATA, 93
- gcf
 - erf.cpp, 202
 - erf.h, 209
- general_Output_parameters
 - ParamStructure, 141
- Get
 - DiffusionCoefficient, 48
 - DiffusionCoefficientsGroup, 67, 68
- get_quads
 - rroots.cpp, 300
 - rroots.h, 304
- GetDerivativeVector
 - MatrixSolver.cpp, 248
- getValue
 - Maths::Interpolation::Linear, 102
- GMRES
 - ITLIN_DATA, 93
- gmres
 - gmres.c, 213
 - MatrixSolver.cpp, 249
- gmres.c, 213
 - gmres, 213
 - gmres_norm2, 214
 - gmres_qrfact, 214
 - gmres_qrsolv, 215
 - itlin_ioctl, 215
 - MAXITER_DEFAULT, 213
- gmres.d, 216
- gmres_norm2
 - gmres.c, 214
 - MatrixSolver.cpp, 250
- gmres_qrfact
 - gmres.c, 214
- gmres_qrsolv
 - gmres.c, 215
- gmres_wrapout
 - MatrixSolver.cpp, 250
 - MatrixSolver.h, 260
- Grid, 75
 - alpha, 79
 - alphaSize, 79
 - epc, 79
 - epcSize, 80
 - Grid, 76
 - Initialize, 77
 - IsInitialized, 78
 - Jacobian, 80
 - L, 80
 - LSize, 80
 - MakeGrid, 78
 - Output, 79
 - pc, 80
 - pcSize, 80
 - type, 80
- Grid.cpp, 217
 - find_alpha, 218
 - find_alpha_res, 218

- maxERR, 218
- Grid.d, 220
- Grid.h, 221
 - find_alpha, 222
- GridElement, 82
 - AllocateMemory, 84
 - GridElement, 83
 - GridElement_parameters, 87
 - Initialize, 84, 85
 - Kfunc, 85
 - operator=, 86
 - SetRegularGridValue, 87
 - size, 87
- GridElement_parameters
 - GridElement, 87
- gser
 - erf.cpp, 203
 - erf.h, 209
- I
 - MatrixSolver.cpp, 247
- i
 - Make_boundary.m, 238
- i_max
 - ITLIN_OPT, 100
- index1d
 - CalculationMatrix, 40
 - Matrix2D, 118
 - Matrix3D, 127
- Initial
 - ITLIN_DATA, 93
- initial_PSD_fileName
 - ParamStructure::PSD, 165
- initial_PSD_inner_psd
 - ParamStructure::PSD, 165
- initial_PSD_J_L7_function
 - ParamStructure::PSD, 165
- initial_PSD_Kp0
 - ParamStructure::PSD, 165
- initial_PSD_outer_psd
 - ParamStructure::PSD, 165
- initial_PSD_some_constant_value
 - ParamStructure::PSD, 166
- initial_PSD_tauSteadyState
 - ParamStructure::PSD, 166
- initial_PSD_Type
 - ParamStructure::PSD, 166
- Initialize
 - BoundaryCondition, 33
 - CalculationMatrix, 40
 - Grid, 77
 - GridElement, 84, 85
 - PSD, 157
 - SourcesAndLosses, 172
- initialized
 - CalculationMatrix, 41
 - Matrix1D, 113
 - Matrix2D, 121
 - Matrix3D, 133
- initialType
 - BoundaryCondition, 36
- int_Daa_loc
 - DiffusionCoefficient.cpp, 189
 - DiffusionCoefficient.h, 197
- int_Dpa_loc
 - DiffusionCoefficient.cpp, 189
 - DiffusionCoefficient.h, 198
- int_Dpp_loc
 - DiffusionCoefficient.cpp, 190
 - DiffusionCoefficient.h, 198
- Intermediate
 - ITLIN_DATA, 93
- Interpolate
 - Matrix1D, 107
 - Matrix2D, 118
 - PSD, 158
- interpolation
 - ParamStructure, 141
- is_active
 - DiffusionCoefficient, 55
- IsInitialized
 - Grid, 78
- iter
 - ITLIN_INFO, 95
- iterfile
 - ITLIN_IO, 97
 - ITLIN_OPT, 100
- iterStep
 - ParamStructure::General_Output_parameters, 73
- itlin.h, 223
 - Absolute, 226
 - CheckEachIter, 227
 - CheckOnRestart, 227
 - CONV_CHECK, 226
 - DATA, 224
 - DATALEVEL, 224
 - daxpy_, 227
 - Debug, 226
 - EPMACH, 224
 - ERROR, 224
 - ERRORLEVEL, 224
 - False, 226
 - FITER, 225
 - FMISC, 225
 - FRES, 225
 - itlin_dataout, 227
 - itlin_noprecon, 227

- itlin_parcheck_and_print, 227
- LOGICAL, 226
- MATVEC, 226
- MAX, 225
- MIN, 225
- Minimum, 226
- MONITOR, 225
- MONITORLEVEL, 225
- None, 226
- PRECON, 226
- PRINT_LEVEL, 226
- RCODE, 225
- Relative, 226
- SIGN, 225
- SMALL, 226
- TERM_CHECK, 226
- True, 226
- Verbose, 226
- zibnum_descale, 228
- zibnum_fwalloc, 228
- zibnum_iwalloc, 228
- zibnum_norm2, 228
- zibnum_pfwalloc, 228
- zibnum_scale, 228
- zibnum_scaled_norm2, 228
- zibnum_scaled_sprod, 229
- ITLIN_DATA
 - Final, 93
 - GBIT, 93
 - GMRES, 93
 - Initial, 93
 - Intermediate, 93
 - PCG, 93
 - Solution, 93
- ITLIN_DATA, 93
 - codeid, 94
 - mode, 94
 - normdx, 94
 - res, 94
 - residuum, 94
 - t, 94
 - tau, 94
- itlin_dataout
 - itlin.h, 227
 - utils.c, 316
- ITLIN_INFO, 95
 - iter, 95
 - nomatvec, 95
 - noprecl, 95
 - noprecon, 95
 - noprecr, 95
 - normdx, 95
 - precision, 96
 - rcode, 96
 - residuum, 96
 - subcode, 96
- ITLIN_IO, 97
 - datfile, 97
 - datlevel, 97
 - errfile, 97
 - errlevel, 97
 - iterfile, 97
 - miscfile, 97
 - monfile, 97
 - monlevel, 98
 - resfile, 98
- itlin_ioctl
 - gmres.c, 215
 - utils.c, 318
- itlin_noprecon
 - itlin.h, 227
 - utils.c, 316
- ITLIN_OPT, 99
 - convcheck, 99
 - datafile, 99
 - datalevel, 99
 - errorfile, 99
 - errorlevel, 100
 - i_max, 100
 - iterfile, 100
 - maxiter, 100
 - miscfile, 100
 - monitorfile, 100
 - monitorlevel, 100
 - rescale, 100
 - resfile, 100
 - rho, 101
 - scale, 101
 - termcheck, 101
 - tol, 101
- itlin_parcheck_and_print
 - itlin.h, 227
 - utils.c, 316
- ITMAX
 - erf.h, 207
- J_L7
 - VF, 25
- J_L7_corrected
 - VF, 25
- jacobi
 - MatrixSolver.cpp, 251
- Jacobian
 - Grid, 80
- Jc2Alpha
 - VF, 26
- Jc_calc
 - VF, 26

- Kfunc
 - GridElement, 85
 - VF, 26
- Kp
 - ParamStructure, 141
- Kp_24_max
 - Make_boundary.m, 237, 238
- L
 - DiffusionCoefficientParamStructure, 60
 - Grid, 80
- L_LowerBoundaryCondition
 - ParamStructure, 141
- L_UpperBoundaryCondition
 - ParamStructure, 142
- lam_max
 - DiffusionCoefficientParamStructure, 60
- lam_min
 - DiffusionCoefficientParamStructure, 60
- Lapack
 - MatrixSolver.cpp, 251
 - MatrixSolver.h, 261
- Linear
 - Maths::Interpolation::Linear, 102
- linear.h, 230
- Linear2D
 - Matrix.cpp, 240
- linearSplineCoef
 - ParamStructure::Interpolation, 91
- load_1d
 - Parameters.cpp, 271
 - Parameters.h, 277
- Load_dxx_parameters
 - DiffusionCoefficientParamStructure, 58
- Load_initial_f
 - PSD, 159
- Load_initial_f_2d
 - PSD, 160
- Load_initial_f_file
 - PSD, 161
- Load_parameters
 - ParamStructure, 139
- LoadBoundaryCondition
 - BoundaryCondition, 33
- LoadDiffusionCoefficient
 - DiffusionCoefficient, 49
- LoadDiffusionCoefficientFromFileWithGrid
 - DiffusionCoefficient, 50
- LoadDiffusionCoefficientFromPlaneFile
 - DiffusionCoefficient, 50
- LoadInitialValue
 - PSD, 161
- loadOrCalculate
 - DiffusionCoefficientParamStructure, 60
- localDiffusionsGrid_alpha
 - ParamStructure, 142
- localDiffusionsGrid_epc
 - ParamStructure, 142
- localDiffusionsGrid_filename
 - ParamStructure, 142
- localDiffusionsGrid_L
 - ParamStructure, 142
- localDiffusionsGrid_pc
 - ParamStructure, 142
- localDiffusionsGrid_type
 - ParamStructure, 142
- log_file
 - Output, 19
- logFileName
 - ParamStructure::General_Output_parameters, 74
- LOGICAL
 - itlin.h, 226
- Lpp
 - ParamStructure, 142
- LSize
 - Grid, 80
- main
 - Main.cpp, 232
- Main.cpp, 231
 - _CRT_SECURE_NO_DEPRECATED, 232
 - check_time, 232
 - main, 232
 - parameters, 235
 - VERB_VERSION_NUMBER, 232
- Main.d, 236
- Make_boundary.m, 237
 - all, 237
 - B_unnorm, 237
 - Bf, 237
 - Bf1, 237
 - Bf2, 237
 - Bf3, 237
 - dat, 238
 - i, 238
 - Kp_24_max, 237, 238
- MakeBoundaryCondition
 - BoundaryCondition, 34
- MakeDLL
 - DiffusionCoefficient, 51
- MakeDLL100
 - DiffusionCoefficient, 51
- MakeDLL_B
 - DiffusionCoefficient, 52
- MakeDLL_BE
 - DiffusionCoefficient, 52
- MakeDLL_BE_res

- DiffusionCoefficient, 52
- MakeDLL_FAKE
 - DiffusionCoefficient, 53
- MakeGrid
 - Grid, 78
- MakeMatrix
 - MatrixSolver.cpp, 252
 - MatrixSolver.h, 261
- MakeModelMatrix_3D
 - MatrixSolver.cpp, 252
 - MatrixSolver.h, 262
- Maths, 15
- Maths::Interpolation, 16
- Maths::Interpolation::Linear, 102
 - ~Linear, 102
 - getValue, 102
 - Linear, 102
 - size, 103
- matrix
 - Matrix.cpp, 240, 241
- Matrix.cpp, 239
 - free_matrix, 240
 - Linear2D, 240
 - matrix, 240, 241
- Matrix.d, 242
- Matrix.h, 243
 - DiagMatrix, 244
- Matrix1D, 104
 - ~Matrix1D, 107
 - AllocateMemory, 107
 - initialized, 113
 - Interpolate, 107
 - Matrix1D, 106
 - max, 108
 - maxabs, 108
 - name, 113
 - operator*, 108
 - operator(), 108
 - operator/, 109
 - operator=, 109
 - Polilinear, 110
 - Polint, 111
 - Ratint, 111
 - readFromFile, 111
 - size_x, 113
 - Spline, 112
 - writeToFile, 112, 113
- Matrix2D, 115
 - ~Matrix2D, 117
 - AllocateMemory, 117
 - index1d, 118
 - initialized, 121
 - Interpolate, 118
 - Matrix2D, 116, 117
 - max, 118
 - name, 121
 - operator*, 118, 119
 - operator/, 119
 - operator=, 119, 120
 - readFromFile, 120
 - size_x, 121
 - size_y, 122
 - writeToFile, 121
- Matrix3D, 123
 - ~Matrix3D, 126
 - abs, 127
 - AllocateMemory, 127
 - change_ind, 133
 - index1d, 127
 - initialized, 133
 - Matrix3D, 125, 126
 - max, 127
 - maxabs, 128
 - name, 133
 - operator*, 128
 - operator(), 128
 - operator+, 128
 - operator-, 129
 - operator/, 129
 - operator=, 129, 130
 - readFromFile, 130, 131
 - size_x, 133
 - size_y, 133
 - size_z, 133
 - Value, 131
 - writeToFile, 131
 - xSlice, 132
 - ySlice, 132
 - zSlice, 132
- MatrixSolver.cpp, 245
 - AddBoundary, 247
 - EE, 247
 - for_norm_A, 248
 - gauss_solve, 248
 - GetDerivativeVector, 248
 - gmres, 249
 - gmres_norm2, 250
 - gmres_wrapout, 250
 - I, 247
 - jacobi, 251
 - Lapack, 251
 - MakeMatrix, 252
 - MakeModelMatrix_3D, 252
 - matvec, 253
 - max2, 253
 - mult_vect2, 253
 - over_relaxation_diag, 253
 - preconr, 254

- SecondDerivativeApproximation_3D, 254
- SolveMatrix, 255
- SOR, 255
- tridag, 255
- MatrixSolver.d, 257
- MatrixSolver.h, 258
 - gauss_solve, 260
 - gmres_wrapout, 260
 - Lapack, 261
 - MakeMatrix, 261
 - MakeModelMatrix_3D, 262
 - over_relaxation_diag, 263
 - SecondDerivativeApproximation_3D, 263
 - SolveMatrix, 264
 - tridag, 264
- MATVEC
 - itlin.h, 226
- matvec
 - MatrixSolver.cpp, 253
- MAX
 - itlin.h, 225
- max
 - Matrix1D, 108
 - Matrix2D, 118
 - Matrix3D, 127
 - ParamStructure::GridElement, 89
 - VF, 26
- max2
 - MatrixSolver.cpp, 253
- max_error
 - bisection.h, 178
- maxabs
 - Matrix1D, 108
 - Matrix3D, 128
- maxERR
 - Grid.cpp, 218
- maxiter
 - ITLIN_OPT, 100
 - rroots.h, 304
- MAXITER_DEFAULT
 - gmres.c, 213
- maxSecondDerivative
 - ParamStructure::Interpolation, 91
- MIN
 - itlin.h, 225
- min
 - ParamStructure::GridElement, 89
- min_Dxx
 - DiffusionCoefficient.cpp, 184
- Minimum
 - itlin.h, 226
- miscfile
 - ITLIN_IO, 97
 - ITLIN_OPT, 100
- MLT_averaging
 - DiffusionCoefficientParamStructure, 60
- mode
 - ITLIN_DATA, 94
- monfile
 - ITLIN_IO, 97
- MONITOR
 - itlin.h, 225
- monitorfile
 - ITLIN_OPT, 100
- MONITORLEVEL
 - itlin.h, 225
- monitorlevel
 - ITLIN_OPT, 100
- monlevel
 - ITLIN_IO, 98
- msg
 - single_error, 169
- mu2pc
 - VF, 27
- mu_calc
 - VF, 27
- mult_vect2
 - MatrixSolver.cpp, 253
- multiplicator
 - DiffusionCoefficientParamStructure, 61
- name
 - Matrix1D, 113
 - Matrix2D, 121
 - Matrix3D, 133
 - ParamStructure::GridElement, 89
- nDays
 - ParamStructure, 142
- nint
 - DiffusionCoefficientParamStructure, 61
- nomatvec
 - ITLIN_INFO, 95
- None
 - itlin.h, 226
- NoNegative
 - ParamStructure, 143
- noprecl
 - ITLIN_INFO, 95
- noprecon
 - ITLIN_INFO, 95
- noprecr
 - ITLIN_INFO, 95
- normdx
 - ITLIN_DATA, 94
 - ITLIN_INFO, 95
- nrrerror
 - erf.cpp, 203
- nu

- DiffusionCoefficientParamStructure, 61
- numberDensity
 - DiffusionCoefficientParamStructure, 61
- omega_lc
 - DiffusionCoefficientParamStructure, 61
- Omega_m
 - DiffusionCoefficientParamStructure, 61
- Omega_mType
 - DiffusionCoefficientParamStructure, 61
- omega_uc
 - DiffusionCoefficientParamStructure, 62
- open_log_file
 - Output, 18
- operator*
 - BoundaryCondition, 34
 - DiffusionCoefficient, 53
 - Matrix1D, 108
 - Matrix2D, 118, 119
 - Matrix3D, 128
- operator()
 - Matrix1D, 108
 - Matrix3D, 128
- operator+
 - DiffusionCoefficient, 54
 - Matrix3D, 128
- operator-
 - Matrix3D, 129
- operator/
 - Matrix1D, 109
 - Matrix2D, 119
 - Matrix3D, 129
- operator=
 - BoundaryCondition, 35
 - DiffusionCoefficient, 54
 - DiffusionCoefficientsGroup, 68, 69
 - GridElement, 86
 - Matrix1D, 109
 - Matrix2D, 119, 120
 - Matrix3D, 129, 130
- Output, 17
 - close_log_file, 17
 - echo, 17
 - Grid, 79
 - log_file, 19
 - open_log_file, 18
 - outputLvl, 19
 - set_output_lvl, 18
- Output.cpp, 266
- Output.d, 267
- Output.h, 268
- output_PSD_fileName4D
 - ParamStructure::PSD, 166
- output_PSD_folderName
 - ParamStructure::PSD, 166
- output_PSD_timeStep
 - ParamStructure::PSD, 166
- Output_without_grid
 - PSD, 162
- output_without_grid_file
 - PSD, 163
- outputLvl
 - Output, 19
 - ParamStructure, 143
- outputModelMatrix
 - ParamStructure, 143
- over_relaxation_diag
 - MatrixSolver.cpp, 253
 - MatrixSolver.h, 263
- parameters
 - Main.cpp, 235
- Parameters.cpp, 270
 - bool2str, 271
 - load_1d, 271
 - ReadFromFile, 271
 - str2bool, 272
 - StrToVal, 272, 273
- Parameters.d, 274
- Parameters.h, 275
 - bool2str, 277
 - DiffusionCoefficientParamStructureList, 277
 - load_1d, 277
 - str2bool, 277
 - StrToVal, 278
- ParamStructure, 135
 - alpha_LowerBoundaryCondition, 140
 - alpha_UpperBoundaryCondition, 140
 - Bf, 140
 - constBf, 140
 - constKp, 140
 - constLpp, 140
 - DLLType, 140
 - DxxParamStructureList, 141
 - fileBf, 141
 - fileKp, 141
 - fileLpp, 141
 - general_Output_parameters, 141
 - interpolation, 141
 - Kp, 141
 - L_LowerBoundaryCondition, 141
 - L_UpperBoundaryCondition, 142
 - Load_parameters, 139
 - localDiffusionsGrid_alpha, 142
 - localDiffusionsGrid_epc, 142
 - localDiffusionsGrid_filename, 142
 - localDiffusionsGrid_L, 142
 - localDiffusionsGrid_pc, 142

- localDiffusionsGrid_type, 142
- Lpp, 142
- nDays, 142
- NoNegative, 143
- outputLvl, 143
- outputModelMatrix, 143
- pc_LowerBoundaryCondition, 143
- pc_UpperBoundaryCondition, 143
- psdLocalDiffusions, 143
- psdRadialDiffusion, 143
- radialDiffusionGrid_alpha, 143
- radialDiffusionGrid_epc, 144
- radialDiffusionGrid_filename, 144
- radialDiffusionGrid_L, 144
- radialDiffusionGrid_pc, 144
- radialDiffusionGrid_type, 144
- sourcesAndLosses, 144
- tau, 144
- tauLpp, 144
- timeStep, 145
- totalIterationsNumber, 145
- useAlphaDifusion, 145
- useBf, 145
- useEnergyAlphaMixedTerms, 145
- useEnergyDifusion, 145
- useKp, 145
- useLpp, 146
- useRadialDifusion, 146
- ParamStructure::BoundaryCondition, 38
 - filename, 38
 - type, 38
 - value, 38
- ParamStructure::General_Output_parameters, 73
 - fileNameID, 73
 - folderName, 73
 - iterStep, 73
 - logFileName, 74
 - timeStep, 74
- ParamStructure::GridElement, 89
 - max, 89
 - min, 89
 - name, 89
 - size, 90
 - useLogScale, 90
- ParamStructure::Interpolation, 91
 - linearSplineCoef, 91
 - maxSecondDerivative, 91
 - type, 91
 - useLog, 91
- ParamStructure::PSD, 164
 - approximationMethod, 165
 - initial_PSD_filename, 165
 - initial_PSD_inner_psd, 165
 - initial_PSD_J_L7_function, 165
 - initial_PSD_Kp0, 165
 - initial_PSD_outer_psd, 165
 - initial_PSD_some_constant_value, 166
 - initial_PSD_tauSteadyState, 166
 - initial_PSD_Type, 166
 - output_PSD_filename4D, 166
 - output_PSD_folderName, 166
 - output_PSD_timeStep, 166
 - SOL_i_max, 167
 - SOL_max_iter_err, 167
 - SOL_maxiter, 167
 - solutionMethod, 167
- ParamStructure::SL, 170
 - SL_E_min, 170
 - SL_E_min_filename, 170
 - SL_L_top, 170
 - SL_L_top_filename, 170
- particle
 - DiffusionCoefficientParamStructure, 62
- pc
 - Grid, 80
- pc2mu
 - VF, 28
- pc_LowerBoundaryCondition
 - ParamStructure, 143
- pc_UpperBoundaryCondition
 - ParamStructure, 143
- PCG
 - ITLIN_DATA, 93
- pcSize
 - Grid, 80
- pfunc
 - VF, 28
- Polilinear
 - Matrix1D, 110
- polilinear
 - polilinear.cpp, 279
 - polilinear.h, 281
- polilinear.cpp, 279
 - polilinear, 279
- polilinear.d, 280
- polilinear.h, 281
 - polilinear, 281
- Polint
 - Matrix1D, 111
- polint
 - polint.cpp, 283
 - polint.h, 286
- polint.cpp, 283
 - polint, 283
- polint.d, 285
- polint.h, 286
 - polint, 286
- precision

- ITLIN_INFO, 96
- PRECON
 - itlin.h, 226
- preconr
 - MatrixSolver.cpp, 254
- PRINT_LEVEL
 - itlin.h, 226
- PSD, 147
 - ~PSD, 151
 - Diffusion_alpha, 151
 - Diffusion_L, 152
 - Diffusion_pc, 153
 - Diffusion_pc_alpha, 154
 - DiffusionMixTermExplicit, 156
 - Initialize, 157
 - Interpolate, 158
 - Load_initial_f, 159
 - Load_initial_f_2d, 160
 - Load_initial_f_file, 161
 - LoadInitialValue, 161
 - Output_without_grid, 162
 - output_without_grid_file, 163
 - PSD, 149, 150
 - PSD_parameters, 163
 - SourcesAndLosses, 163
- PSD.cpp, 288
 - checkInf, 289
 - steady_state, 289
- PSD.d, 291
- PSD.h, 292
 - steady_state, 293
- PSD_parameters
 - PSD, 163
- psdLocalDiffusions
 - ParamStructure, 143
- psdRadialDiffusion
 - ParamStructure, 143
- quad1
 - DiffusionCoefficient.cpp, 191
- radialDiffusionGrid_alpha
 - ParamStructure, 143
- radialDiffusionGrid_epc
 - ParamStructure, 144
- radialDiffusionGrid_filename
 - ParamStructure, 144
- radialDiffusionGrid_L
 - ParamStructure, 144
- radialDiffusionGrid_pc
 - ParamStructure, 144
- radialDiffusionGrid_type
 - ParamStructure, 144
- Ratint
 - Matrix1D, 111
- ratint
 - ratint.cpp, 294
 - ratint.h, 297
- ratint.cpp, 294
 - ratint, 294
 - TINY, 294
- ratint.d, 296
- ratint.h, 297
 - ratint, 297
- RCODE
 - itlin.h, 225
- rcode
 - ITLIN_INFO, 96
- ReadFromFile
 - Parameters.cpp, 271
- readFromFile
 - Matrix1D, 111
 - Matrix2D, 120
 - Matrix3D, 130, 131
- recurse
 - rroots.cpp, 300
- Relative
 - itlin.h, 226
- res
 - ITLIN_DATA, 94
- rescale
 - ITLIN_OPT, 100
- resfile
 - ITLIN_IO, 98
 - ITLIN_OPT, 100
- residuum
 - ITLIN_DATA, 94
 - ITLIN_INFO, 96
- rho
 - ITLIN_OPT, 101
- roots
 - rroots.cpp, 301
 - rroots.h, 305
- rroots.cpp, 298
 - deflate, 299
 - diff_poly, 299
 - find_quad, 299
 - get_quads, 300
 - recurse, 300
 - roots, 301
- rroots.d, 302
- rroots.h, 303
 - DBL_EPSILON, 304
 - get_quads, 304
 - maxiter, 304
 - roots, 305
- rrouts
 - DiffusionCoefficient.cpp, 191

- DiffusionCoefficient.h, 199
- s
 - DiffusionCoefficientParamStructure, 62
- Scale
 - DiffusionCoefficient, 54
- scale
 - ITLIN_OPT, 101
- SecondDerivativeApproximation_3D
 - MatrixSolver.cpp, 254
 - MatrixSolver.h, 263
- set_output_lvl
 - Output, 18
- SetRegularGridValue
 - GridElement, 87
- SIGN
 - itlin.h, 225
- single_error, 168
 - code, 168
 - msg, 169
 - single_error, 168
 - single_error, 168
- size
 - GridElement, 87
 - Maths::Interpolation::Linear, 103
 - ParamStructure::GridElement, 90
- size_x
 - Matrix1D, 113
 - Matrix2D, 121
 - Matrix3D, 133
- size_y
 - Matrix2D, 122
 - Matrix3D, 133
- size_z
 - Matrix3D, 133
- SL_E_min
 - ParamStructure::SL, 170
- SL_E_min_filename
 - ParamStructure::SL, 170
- SL_L_top
 - ParamStructure::SL, 170
- SL_L_top_filename
 - ParamStructure::SL, 170
- SL_parameters
 - SourcesAndLosses, 172
- SMALL
 - itlin.h, 226
- SOL_i_max
 - ParamStructure::PSD, 167
- SOL_max_iter_err
 - ParamStructure::PSD, 167
- SOL_maxiter
 - ParamStructure::PSD, 167
- Solution
 - ITLIN_DATA, 93
- solutionMethod
 - ParamStructure::PSD, 167
- SolveMatrix
 - MatrixSolver.cpp, 255
 - MatrixSolver.h, 264
- SOR
 - MatrixSolver.cpp, 255
- SourcesAndLosses, 171
 - Initialize, 172
 - PSD, 163
 - SL_parameters, 172
 - SourcesAndLosses, 171
- sourcesAndLosses
 - ParamStructure, 144
- SourcesAndLosses.cpp, 306
 - err, 306
- SourcesAndLosses.d, 307
- SourcesAndLosses.h, 308
- Spline
 - Matrix1D, 112
- spline
 - spline.cpp, 309
 - spline.h, 312
- spline.cpp, 309
 - spline, 309
 - splint, 310
- spline.d, 311
- spline.h, 312
 - spline, 312
 - splint, 313
- splint
 - spline.cpp, 310
 - spline.h, 313
- steady_state
 - PSD.cpp, 289
 - PSD.h, 293
- str2bool
 - erf.cpp, 204
 - Parameters.cpp, 272
 - Parameters.h, 277
- StrToVal
 - Parameters.cpp, 272, 273
 - Parameters.h, 278
- subcode
 - ITLIN_INFO, 96
- t
 - ITLIN_DATA, 94
- tau
 - ITLIN_DATA, 94
 - ParamStructure, 144
- tauLpp
 - ParamStructure, 144

- TERM_CHECK
 - itlin.h, 226
- termcheck
 - ITLIN_OPT, 101
- time_end
 - DiffusionCoefficientParamStructure, 62
- time_start
 - DiffusionCoefficientParamStructure, 62
- timeStep
 - ParamStructure, 145
 - ParamStructure::General_Output_parameters, 74
- TINY
 - ratint.cpp, 294
- tol
 - ITLIN_OPT, 101
- TOLMAX
 - utils.c, 315
- TOLMIN
 - utils.c, 315
- total_size
 - CalculationMatrix, 41
- totalIterationsNumber
 - ParamStructure, 145
- tridag
 - MatrixSolver.cpp, 255
 - MatrixSolver.h, 264
- True
 - itlin.h, 226
- type
 - DiffusionCoefficient, 55
 - Grid, 80
 - ParamStructure::BoundaryCondition, 38
 - ParamStructure::Interpolation, 91
- Update
 - BoundaryCondition, 35
- useAlphaDifusion
 - ParamStructure, 145
- useBf
 - ParamStructure, 145
- useEnergyAlphaMixedTerms
 - ParamStructure, 145
- useEnergyDifusion
 - ParamStructure, 145
- useKp
 - ParamStructure, 145
- useLog
 - ParamStructure::Interpolation, 91
- useLogScale
 - ParamStructure::GridElement, 90
- useLpp
 - ParamStructure, 146
- useRadialDifusion
 - ParamStructure, 146
- useScale
 - DiffusionCoefficient, 55
 - DiffusionCoefficientParamStructure, 62
- utils.c, 315
 - itlin_dataout, 316
 - itlin_ioctl, 318
 - itlin_noprecon, 316
 - itlin_parcheck_and_print, 316
 - TOLMAX, 315
 - TOLMIN, 315
 - zibnum_descale, 316
 - zibnum_fwalloc, 316
 - zibnum_iwalloc, 317
 - zibnum_norm2, 317
 - zibnum_pfwalloc, 317
 - zibnum_scale, 317
 - zibnum_scaled_norm2, 317
 - zibnum_scaled_sprod, 317
- utils.d, 319
- Value
 - Matrix3D, 131
- value
 - BoundaryCondition, 36
 - ParamStructure::BoundaryCondition, 38
- variousConstants.h, 320
- variousFunctions.cpp, 321
- variousFunctions.d, 323
- variousFunctions.h, 324
- VC, 20
- VERB_VERSION_NUMBER
 - Main.cpp, 232
- Verbose
 - itlin.h, 226
- VF, 21
 - alc, 22
 - Alpha2J, 22
 - Alpha2Jc, 23
 - B, 23
 - bounce_time, 23
 - density, 24
 - Df, 24
 - dtostr, 24
 - f_interp, 24
 - G, 25
 - J_L7, 25
 - J_L7_corrected, 25
 - Jc2Alpha, 26
 - Jc_calc, 26
 - Kfunc, 26
 - max, 26
 - mu2pc, 27
 - mu_calc, 27

- pc2mu, [28](#)
- pfunc, [28](#)
- Y, [28](#)
- Y_old, [29](#)
- waveName
 - DiffusionCoefficientParamStructure, [62](#)
- waveType
 - DiffusionCoefficientParamStructure, [63](#)
- what
 - error_msg, [72](#)
- writeToFile
 - CalculationMatrix, [41](#)
 - Matrix1D, [112](#), [113](#)
 - Matrix2D, [121](#)
 - Matrix3D, [131](#)
- x_size
 - CalculationMatrix, [41](#)
- xSlice
 - Matrix3D, [132](#)
- Y
 - VF, [28](#)
- Y_old
 - VF, [29](#)
- y_size
 - CalculationMatrix, [41](#)
- ySlice
 - Matrix3D, [132](#)
- zibnum_descale
 - itlin.h, [228](#)
 - utils.c, [316](#)
- zibnum_fwalloc
 - itlin.h, [228](#)
 - utils.c, [316](#)
- zibnum_iwalloc
 - itlin.h, [228](#)
 - utils.c, [317](#)
- zibnum_norm2
 - itlin.h, [228](#)
 - utils.c, [317](#)
- zibnum_pfalloc
 - itlin.h, [228](#)
 - utils.c, [317](#)
- zibnum_scale
 - itlin.h, [228](#)
 - utils.c, [317](#)
- zibnum_scaled_norm2
 - itlin.h, [228](#)
 - utils.c, [317](#)
- zibnum_scaled_sprod
 - itlin.h, [229](#)
- utils.c, [317](#)
- zSlice
 - Matrix3D, [132](#)