

# libQMC2 Documentation

Generated by Doxygen 1.8.3.1

Wed Sep 4 2013 16:50:59



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List	3
<b>3</b>	<b>Class Documentation</b>	<b>7</b>
3.1	AlphaHarmonicOscillator Class Reference	7
3.1.1	Detailed Description	8
3.1.2	Member Function Documentation	8
3.1.2.1	get_coulomb_element	8
3.1.2.2	get_dell_alpha_phi	8
3.1.2.3	get_parameter	8
3.1.2.4	get_qnums	8
3.1.2.5	H	8
3.1.2.6	set_parameter	9
3.1.2.7	set_qnum_indie_terms	9
3.2	ASGD Class Reference	9
3.2.1	Detailed Description	11
3.2.2	Member Function Documentation	11
3.2.2.1	get_total_grad	11
3.2.2.2	get_variational_gradients	11
3.3	AtomCore Class Reference	11
3.3.1	Detailed Description	12
3.3.2	Member Function Documentation	12
3.3.2.1	get_pot_E	12
3.4	BasisFunctions Class Reference	12
3.4.1	Detailed Description	12
3.4.2	Member Function Documentation	13
3.4.2.1	eval	13
3.5	Blocking Class Reference	13
3.5.1	Member Function Documentation	14

3.5.1.1	<a href="#">block_data</a>	14
3.5.1.2	<a href="#">get_initial_error</a>	14
3.5.1.3	<a href="#">get_unique_blocks</a>	14
3.6	<a href="#">Bosons Class Reference</a>	14
3.6.1	<a href="#">Detailed Description</a>	15
3.6.2	<a href="#">Member Function Documentation</a>	15
3.6.2.1	<a href="#">calc_for_newpos</a>	15
3.6.2.2	<a href="#">copy_walker</a>	15
3.6.2.3	<a href="#">get_spatial_grad</a>	15
3.6.2.4	<a href="#">get_spatial_wf</a>	15
3.6.2.5	<a href="#">initialize</a>	16
3.6.2.6	<a href="#">reset_walker</a>	16
3.6.2.7	<a href="#">update_walker</a>	16
3.7	<a href="#">Brute_Force Class Reference</a>	16
3.7.1	<a href="#">Detailed Description</a>	17
3.7.2	<a href="#">Member Function Documentation</a>	17
3.7.2.1	<a href="#">copy_walker</a>	17
3.7.2.2	<a href="#">get_necessities</a>	17
3.7.2.3	<a href="#">reset_walker</a>	17
3.7.2.4	<a href="#">update_walker</a>	17
3.8	<a href="#">Coulomb Class Reference</a>	17
3.8.1	<a href="#">Detailed Description</a>	18
3.8.2	<a href="#">Member Function Documentation</a>	18
3.8.2.1	<a href="#">get_pot_E</a>	18
3.9	<a href="#">DiAtomCore Class Reference</a>	18
3.9.1	<a href="#">Member Function Documentation</a>	19
3.9.1.1	<a href="#">get_pot_E</a>	19
3.10	<a href="#">Diffusion Class Reference</a>	19
3.10.1	<a href="#">Detailed Description</a>	20
3.10.2	<a href="#">Member Function Documentation</a>	20
3.10.2.1	<a href="#">call_RNG</a>	20
3.10.2.2	<a href="#">get_g_ratio</a>	20
3.10.2.3	<a href="#">get_new_pos</a>	21
3.10.2.4	<a href="#">set_dt</a>	21
3.11	<a href="#">Distribution Class Reference</a>	21
3.11.1	<a href="#">Detailed Description</a>	22
3.11.2	<a href="#">Constructor &amp; Destructor Documentation</a>	22
3.11.2.1	<a href="#">Distribution</a>	22
3.11.3	<a href="#">Member Function Documentation</a>	22
3.11.3.1	<a href="#">dump</a>	22

3.11.3.2	finalize	22
3.11.3.3	rerun	22
3.12	DiTransform Class Reference	23
3.12.1	Member Function Documentation	23
3.12.1.1	del_phi	23
3.12.1.2	get_parameter	24
3.12.1.3	lapl_phi	24
3.12.1.4	phi	24
3.12.1.5	set_parameter	24
3.12.1.6	set_qnum_indie_terms	24
3.13	DMC Class Reference	24
3.13.1	Detailed Description	26
3.13.2	Member Function Documentation	26
3.13.2.1	Evolve_walker	26
3.13.2.2	iterate_walker	26
3.13.2.3	move_authorized	27
3.13.2.4	node_comm	27
3.13.2.5	save_distribution	27
3.13.2.6	set_trial_positions	27
3.13.2.7	switch_souls	27
3.13.3	Member Data Documentation	27
3.13.3.1	K	27
3.14	DMCparams Struct Reference	28
3.14.1	Detailed Description	28
3.15	DoubleWell Class Reference	28
3.15.1	Member Function Documentation	29
3.15.1.1	get_pot_E	29
3.16	ErrorEstimator Class Reference	29
3.16.1	Detailed Description	30
3.16.2	Constructor & Destructor Documentation	30
3.16.2.1	ErrorEstimator	30
3.16.3	Member Function Documentation	30
3.16.3.1	combine_mean	31
3.16.3.2	combine_variance	31
3.16.3.3	finalize	31
3.16.3.4	init_file	31
3.16.3.5	node_comm_gather_data	31
3.16.3.6	node_comm_scatter_data	31
3.16.3.7	update_data	31
3.17	ExpandedBasis Class Reference	32

3.17.1	Member Function Documentation	32
3.17.1.1	del_phi	32
3.17.1.2	lapl_phi	32
3.17.1.3	phi	33
3.17.1.4	set_qnum_indie_terms	33
3.18	Fermions Class Reference	33
3.18.1	Detailed Description	34
3.18.2	Member Function Documentation	34
3.18.2.1	calc_for_newpos	34
3.18.2.2	copy_walker	34
3.18.2.3	get_spatial_grad	34
3.18.2.4	get_spatial_wf	35
3.18.2.5	initialize	35
3.18.2.6	make_merged_inv	35
3.18.2.7	reset_walker	35
3.18.2.8	update_walker	35
3.18.3	Member Data Documentation	35
3.18.3.1	I	35
3.19	Fokker_Planck Class Reference	35
3.19.1	Detailed Description	36
3.19.2	Member Function Documentation	36
3.19.2.1	get_g_ratio	36
3.19.2.2	get_new_pos	36
3.20	GeneralParams Struct Reference	37
3.20.1	Detailed Description	37
3.20.2	Member Data Documentation	37
3.20.2.1	systemConstant	37
3.21	Harmonic_osc Class Reference	38
3.21.1	Detailed Description	38
3.21.2	Member Function Documentation	38
3.21.2.1	get_pot_E	38
3.22	HartreeFock Class Reference	38
3.23	hydrogenicOrbitals Class Reference	39
3.23.1	Detailed Description	40
3.23.2	Member Function Documentation	40
3.23.2.1	get_coulomb_element	40
3.23.2.2	get_dell_alpha_phi	40
3.23.2.3	get_parameter	40
3.23.2.4	set_parameter	40
3.23.2.5	set_qnum_indie_terms	41

3.24 Importance Class Reference	41
3.24.1 Detailed Description	41
3.24.2 Member Function Documentation	41
3.24.2.1 calculate_energy_necessities	41
3.24.2.2 copy_walker	42
3.24.2.3 get_necessities	42
3.24.2.4 reset_walker	42
3.24.2.5 update_necessities	42
3.24.2.6 update_walker	42
3.25 Jastrow Class Reference	42
3.25.1 Detailed Description	43
3.25.2 Member Function Documentation	44
3.25.2.1 get_derivative_num	44
3.25.2.2 get_dJ_matrix	44
3.25.2.3 get_dJ_matrix	44
3.25.2.4 get_grad	44
3.25.2.5 get_grad	44
3.25.2.6 get_j_ratio	44
3.25.2.7 get_lapl_sum	45
3.25.2.8 get_laplaciansum_num	45
3.25.2.9 get_parameter	45
3.25.2.10 get_val	45
3.25.2.11 get_variational_derivative	45
3.25.2.12 initialize	45
3.25.2.13 set_parameter	46
3.26 Minimizer Class Reference	46
3.26.1 Detailed Description	47
3.26.2 Constructor & Destructor Documentation	47
3.26.2.1 Minimizer	47
3.26.3 Member Function Documentation	47
3.26.3.1 update_parameters	47
3.27 MinimizerParams Struct Reference	47
3.27.1 Detailed Description	48
3.28 No_Jastrow Class Reference	48
3.28.1 Detailed Description	49
3.28.2 Member Function Documentation	49
3.28.2.1 get_dJ_matrix	49
3.28.2.2 get_grad	49
3.28.2.3 get_grad	49
3.28.2.4 get_j_ratio	49

3.28.2.5	<a href="#">get_lapl_sum</a>	50
3.28.2.6	<a href="#">get_parameter</a>	50
3.28.2.7	<a href="#">get_val</a>	50
3.28.2.8	<a href="#">get_variational_derivative</a>	50
3.28.2.9	<a href="#">initialize</a>	50
3.28.2.10	<a href="#">set_parameter</a>	51
3.29	<a href="#">NO_STDOUT Class Reference</a>	51
3.29.1	<a href="#">Detailed Description</a>	51
3.30	<a href="#">Orbitals Class Reference</a>	51
3.30.1	<a href="#">Detailed Description</a>	53
3.30.2	<a href="#">Member Function Documentation</a>	53
3.30.2.1	<a href="#">del_phi</a>	53
3.30.2.2	<a href="#">get_coulomb_element</a>	53
3.30.2.3	<a href="#">get_parameter</a>	53
3.30.2.4	<a href="#">get_variational_derivative</a>	53
3.30.2.5	<a href="#">lapl_phi</a>	54
3.30.2.6	<a href="#">num_ddiff</a>	54
3.30.2.7	<a href="#">num_diff</a>	54
3.30.2.8	<a href="#">phi</a>	54
3.30.2.9	<a href="#">set_parameter</a>	54
3.30.2.10	<a href="#">set_qnum_indie_terms</a>	55
3.30.2.11	<a href="#">testDell</a>	55
3.30.2.12	<a href="#">testLaplace</a>	55
3.31	<a href="#">OutputHandler Class Reference</a>	55
3.31.1	<a href="#">Detailed Description</a>	56
3.31.2	<a href="#">Constructor &amp; Destructor Documentation</a>	56
3.31.2.1	<a href="#">OutputHandler</a>	56
3.31.3	<a href="#">Member Function Documentation</a>	56
3.31.3.1	<a href="#">dump</a>	56
3.31.3.2	<a href="#">finalize</a>	56
3.31.3.3	<a href="#">init_file</a>	57
3.32	<a href="#">Pade_Jastrow Class Reference</a>	57
3.32.1	<a href="#">Detailed Description</a>	58
3.32.2	<a href="#">Member Function Documentation</a>	58
3.32.2.1	<a href="#">get_dJ_matrix</a>	58
3.32.2.2	<a href="#">get_grad</a>	58
3.32.2.3	<a href="#">get_grad</a>	58
3.32.2.4	<a href="#">get_j_ratio</a>	58
3.32.2.5	<a href="#">get_lapl_sum</a>	59
3.32.2.6	<a href="#">get_parameter</a>	59



3.32.2.7	<a href="#">get_val</a>	59
3.32.2.8	<a href="#">get_variational_derivative</a>	59
3.32.2.9	<a href="#">initialize</a>	59
3.32.2.10	<a href="#">set_parameter</a>	59
3.33	<a href="#">ParParams Struct Reference</a>	60
3.33.1	<a href="#">Detailed Description</a>	60
3.34	<a href="#">Potential Class Reference</a>	60
3.34.1	<a href="#">Detailed Description</a>	61
3.34.2	<a href="#">Member Function Documentation</a>	61
3.34.2.1	<a href="#">get_pot_E</a>	61
3.35	<a href="#">QMC Class Reference</a>	61
3.35.1	<a href="#">Detailed Description</a>	64
3.35.2	<a href="#">Constructor &amp; Destructor Documentation</a>	64
3.35.2.1	<a href="#">QMC</a>	64
3.35.3	<a href="#">Member Function Documentation</a>	64
3.35.3.1	<a href="#">calculate_energy_necessities</a>	64
3.35.3.2	<a href="#">calculate_local_energy</a>	64
3.35.3.3	<a href="#">copy_walker</a>	64
3.35.3.4	<a href="#">diffuse_walker</a>	65
3.35.3.5	<a href="#">estimate_error</a>	65
3.35.3.6	<a href="#">get_gradients</a>	65
3.35.3.7	<a href="#">get_gradients</a>	65
3.35.3.8	<a href="#">get_lapsum</a>	65
3.35.3.9	<a href="#">get_wf_value</a>	65
3.35.3.10	<a href="#">metropolis_test</a>	65
3.35.3.11	<a href="#">move_authorized</a>	66
3.35.3.12	<a href="#">reset_walker</a>	66
3.35.3.13	<a href="#">save_distribution</a>	66
3.35.3.14	<a href="#">set_spin_state</a>	66
3.35.3.15	<a href="#">set_trial_positions</a>	66
3.35.3.16	<a href="#">test_gradients</a>	67
3.35.3.17	<a href="#">test_ratios</a>	67
3.35.3.18	<a href="#">update_walker</a>	67
3.35.4	<a href="#">Member Data Documentation</a>	67
3.35.4.1	<a href="#">cycle</a>	67
3.35.4.2	<a href="#">n_w</a>	67
3.36	<a href="#">Sampler Class Reference</a>	67
3.37	<a href="#">Sampling Class Reference</a>	68
3.37.1	<a href="#">Member Function Documentation</a>	69
3.37.1.1	<a href="#">call_RNG</a>	69

3.37.1.2	<a href="#">copy_walker</a>	69
3.37.1.3	<a href="#">get_branching_Gfunc</a>	69
3.37.1.4	<a href="#">get_g_ratio</a>	69
3.37.1.5	<a href="#">get_necessities</a>	70
3.37.1.6	<a href="#">reset_walker</a>	70
3.37.1.7	<a href="#">set_spin_state</a>	70
3.37.1.8	<a href="#">update_pos</a>	70
3.37.1.9	<a href="#">update_walker</a>	70
3.37.2	<a href="#">Member Data Documentation</a>	70
3.37.2.1	<a href="#">diffusion</a>	70
3.37.2.2	<a href="#">end</a>	70
3.37.2.3	<a href="#">start</a>	71
3.38	<a href="#">Simple Class Reference</a>	71
3.38.1	<a href="#">Detailed Description</a>	71
3.38.2	<a href="#">Member Function Documentation</a>	71
3.38.2.1	<a href="#">get_g_ratio</a>	71
3.38.2.2	<a href="#">get_new_pos</a>	72
3.39	<a href="#">SimpleVar Class Reference</a>	72
3.39.1	<a href="#">Detailed Description</a>	72
3.39.2	<a href="#">Member Function Documentation</a>	73
3.39.2.1	<a href="#">update_data</a>	73
3.40	<a href="#">STDOUT Class Reference</a>	73
3.40.1	<a href="#">Detailed Description</a>	73
3.41	<a href="#">stdoutASGD Class Reference</a>	73
3.41.1	<a href="#">Detailed Description</a>	74
3.41.2	<a href="#">Member Function Documentation</a>	74
3.41.2.1	<a href="#">dump</a>	74
3.42	<a href="#">stdoutDMC Class Reference</a>	74
3.42.1	<a href="#">Detailed Description</a>	75
3.42.2	<a href="#">Member Function Documentation</a>	75
3.42.2.1	<a href="#">dump</a>	75
3.43	<a href="#">System Class Reference</a>	75
3.43.1	<a href="#">Detailed Description</a>	76
3.43.2	<a href="#">Member Function Documentation</a>	76
3.43.2.1	<a href="#">calc_for_newpos</a>	76
3.43.2.2	<a href="#">copy_walker</a>	76
3.43.2.3	<a href="#">get_potential_energy</a>	77
3.43.2.4	<a href="#">get_spatial_grad</a>	77
3.43.2.5	<a href="#">initialize</a>	77
3.43.2.6	<a href="#">reset_walker</a>	77

3.43.2.7	set_spin_state	77
3.43.2.8	update_walker	77
3.43.3	Member Data Documentation	77
3.43.3.1	end	77
3.43.3.2	start	78
3.44	SystemObjects Struct Reference	78
3.44.1	Detailed Description	78
3.45	VariationalParams Struct Reference	78
3.45.1	Detailed Description	79
3.46	VMC Class Reference	79
3.46.1	Detailed Description	80
3.46.2	Member Function Documentation	80
3.46.2.1	move_authorized	80
3.46.2.2	save_distribution	80
3.46.2.3	set_trial_positions	80
3.46.2.4	store_walkers	80
3.47	VMCparams Struct Reference	80
3.47.1	Detailed Description	81
3.48	Walker Class Reference	81
3.48.1	Detailed Description	82
3.48.2	Constructor & Destructor Documentation	83
3.48.2.1	Walker	83
3.48.3	Member Function Documentation	83
3.48.3.1	calc_r_i	83
3.48.3.2	calc_r_i2	83
3.48.3.3	calc_r_rel	83
3.48.3.4	get_r_i	83
3.48.3.5	get_r_i2	83
3.48.3.6	kill	84
3.48.3.7	make_rel_matrix	84
3.48.3.8	print	84
3.48.3.9	recv_soul	84
3.48.3.10	ressurect	84
3.48.3.11	send_soul	84



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

BasisFunctions . . . . .	12
Diffusion . . . . .	19
Fokker_Planck . . . . .	35
Simple . . . . .	71
DMCparams . . . . .	28
ErrorEstimator . . . . .	29
Blocking . . . . .	13
SimpleVar . . . . .	72
GeneralParams . . . . .	37
HartreeFock . . . . .	38
Jastrow . . . . .	42
No_Jastrow . . . . .	48
Pade_Jastrow . . . . .	57
Minimizer . . . . .	46
ASGD . . . . .	9
MinimizerParams . . . . .	47
Orbitals . . . . .	51
AlphaHarmonicOscillator . . . . .	7
DiTransform . . . . .	23
ExpandedBasis . . . . .	32
hydrogenicOrbitals . . . . .	39
OutputHandler . . . . .	55
Distribution . . . . .	21
stdoutASGD . . . . .	73
stdoutDMC . . . . .	74
ParParams . . . . .	60
Potential . . . . .	60
AtomCore . . . . .	11
Coulomb . . . . .	17
DiAtomCore . . . . .	18
DoubleWell . . . . .	28
Harmonic_osc . . . . .	38
QMC . . . . .	61
DMC . . . . .	24
VMC . . . . .	79
Sampler . . . . .	67

Sampling . . . . .	68
Brute_Force . . . . .	16
Importance . . . . .	41
STDOUT . . . . .	73
NO_STDOUT . . . . .	51
System . . . . .	75
Bosons . . . . .	14
Fermions . . . . .	33
SystemObjects . . . . .	78
VariationalParams . . . . .	78
VMCparams . . . . .	80
Walker . . . . .	81

## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">AlphaHarmonicOscillator</a>	Harmonic Oscillator single particle wave function class. Uses the HarmonicOscillator Basis-Function subclasses auto-generated by SymPy through the orbitalsGenerator tool . . . . .	7
<a href="#">ASGD</a>	Implementation for the Adaptive Stochastic Gradient Descent method ( <a href="#">ASGD</a> ) Used to find optimal variational parameters using adaptive step lengths . . . . .	9
<a href="#">AtomCore</a>	Implementation of the Atom Core potential. $-Z/r$ . . . . .	11
<a href="#">BasisFunctions</a>	The Superclass shell for orbital basis functions . . . . .	12
<a href="#">Blocking</a>	. . . . .	13
<a href="#">Bosons</a>	The Boson system class . . . . .	14
<a href="#">Brute_Force</a>	Implementation of the Brute Force <a href="#">QMC</a> . Uses the Simple diffusion class. All methods are empty except for the energy necessities part which requires the gradients to be calculated (not using the Quantum Force) . . . . .	16
<a href="#">Coulomb</a>	Implementation of the <a href="#">Coulomb</a> potential. $1/r_{ij}$ . . . . .	17
<a href="#">DiAtomCore</a>	. . . . .	18
<a href="#">Diffusion</a>	Class containing rules for walker movement based on diffusion models. Serves as class member in the <a href="#">Sampling</a> class. Brute force implies the <a href="#">Simple</a> diffusion model, while <a href="#">Importance Sampling</a> implies the Fokker Planck diffusion . . . . .	19
<a href="#">Distribution</a>	Class for calculating distribution functions such as the one-body density. Does not collect data itself, but works merely as a control organ for the <a href="#">QMC</a> class, calling it's methods for storing position data . . . . .	21
<a href="#">DiTransform</a>	. . . . .	23
<a href="#">DMC</a>	Implementation of the <a href="#">Diffusion</a> Monte-Carlo Method. Very little needs to be added when the <a href="#">QMC</a> superclass holds all the general functionality . . . . .	24
<a href="#">DMCparams</a>	Struct used to initialize <a href="#">DMC</a> parameters . . . . .	28
<a href="#">DoubleWell</a>	. . . . .	28

<a href="#">ErrorEstimator</a>	
Class handling error estimations of the <a href="#">QMC</a> methods. The <a href="#">QMC</a> class holds an object of this type, calling the <code>update_data</code> function in order to update the sampling pool. <code>finalize()</code> then either dumps the samples to file for later processing, or calculates an estimate . . . . .	29
<a href="#">ExpandedBasis</a> . . . . .	32
<a href="#">Fermions</a>	
The Fermion system class . . . . .	33
<a href="#">Fokker_Planck</a>	
Anisotropic diffusion by the Fokker-Planck equation . . . . .	35
<a href="#">GeneralParams</a>	
Struct used to initialize general parameters . . . . .	37
<a href="#">Harmonic_osc</a>	
Implementation of the Harmonic Oscillator potential. $0.5*w**2*r**2$ . . . . .	38
<a href="#">HartreeFock</a> . . . . .	38
<a href="#">hydrogenicOrbitals</a>	
Hydrogen-like single particle wave function class. Uses the hydrogenic BasisFunction subclasses auto-generated by SymPy through the orbitalsGenerator tool . . . . .	39
<a href="#">Importance</a>	
Implementation of <a href="#">Importance</a> sampled <a href="#">QMC</a> . Using the Fokker-Planck diffusion class. Introduces the Quantum Force . . . . .	41
<a href="#">Jastrow</a>	
The class representing the <a href="#">Jastrow</a> correlation functions Holds all data concerning the <a href="#">Jastrow</a> function and it's influence on the <a href="#">QMC</a> algorithm . . . . .	42
<a href="#">Minimizer</a>	
Class for minimization methods used to obtain optimal variational parameters . . . . .	46
<a href="#">MinimizerParams</a>	
Struct used to initialize Minimization parameters . . . . .	47
<a href="#">No_Jastrow</a>	
Class loaded when no correlation factor is used . . . . .	48
<a href="#">NO_STDOUT</a>	
Class for suppressing standard output. Every node but the master has this. If-tests around <code>cout</code> is avoided . . . . .	51
<a href="#">Orbitals</a>	
Superclass for the single particle orbital classes. Handles everything specific regarding choice of single particle basis . . . . .	51
<a href="#">OutputHandler</a>	
Class for handling output-methods. Designed to avoid rewriting code, as well as avoid if-tests if output is not desired . . . . .	55
<a href="#">Pade_Jastrow</a>	
The Pade <a href="#">Jastrow</a> factor with a single variational parameter . . . . .	57
<a href="#">ParParams</a>	
Struct used to initialize parallelization parameters . . . . .	60
<a href="#">Potential</a>	
Superclass for potentials. Potentials are stores in a vector in the system object . . . . .	60
<a href="#">QMC</a>	
The <a href="#">QMC</a> superclass. Holds implementations of general functions for both <a href="#">VMC</a> and <a href="#">DMC</a> in order to avoid rewriting code and emphasize the similarities . . . . .	61
<a href="#">Sampler</a> . . . . .	67
<a href="#">Sampling</a> . . . . .	68
<a href="#">Simple</a>	
<a href="#">Simple</a> Isotropic diffusion model . . . . .	71
<a href="#">SimpleVar</a>	
Calculates the simple variance of the sampled values . . . . .	72
<a href="#">STDOUT</a>	
Class for handling standard output. Only the master node has this object . . . . .	73
<a href="#">stdoutASGD</a>	
Class for handling the output of <a href="#">ASGD</a> . Outputs values such as the variational gradients, step length, variational parameters, etc . . . . .	73



<a href="#">stdoutDMC</a>	Class for handling the output of <a href="#">DMC</a> . Outputs values such as the trial energy, dmc energy, number of walkers, etc . . . . .	74
<a href="#">System</a>	The system class separating <a href="#">Fermions</a> and <a href="#">Bosons</a> . Designed to generalize the solver in terms of particle species . . . . .	75
<a href="#">SystemObjects</a>	Struct used to initialize system objects . . . . .	78
<a href="#">VariationalParams</a>	Struct used to initialize the varational parameters . . . . .	78
<a href="#">VMC</a>	Implementation of the Variational Monte-Carlo Method. Very little needs to be added when the <a href="#">QMC</a> superclass holds all the general functionality . . . . .	79
<a href="#">VMCparams</a>	Struct used to initialize <a href="#">VMC</a> parameters . . . . .	80
<a href="#">Walker</a>	Class representing a Random <a href="#">Walker</a> . Holds position data, alive state, etc. Designed to lighten function arguments, and ease implementation of <a href="#">QMC</a> methods involving multiple walkers. Alot of values are stored to avoid calculating the same value twice . . . . .	81



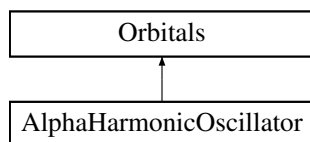
## Chapter 3

# Class Documentation

### 3.1 AlphaHarmonicOscillator Class Reference

Harmonic Oscillator single particle wave function class. Uses the HarmonicOscillator BasisFunction subclasses auto-generated by SymPy through the orbitalsGenerator tool.

Inheritance diagram for AlphaHarmonicOscillator:



#### Public Member Functions

- **AlphaHarmonicOscillator** ([GeneralParams](#) &, [VariationalParams](#) &)
- void [set\\_qnum\\_indie\\_terms](#) ([Walker](#) \*walker, int i)

#### Protected Member Functions

- double [get\\_dell\\_alpha\\_phi](#) ([Walker](#) \*walker, int p, int q\_num)
- void [get\\_qnums](#) ()
- void **get\_qnums3D** ()
- void **setup\_basis** ()
- void **setup\_basis3D** ()
- double [get\\_coulomb\\_element](#) (const arma::uvec &qnum\_set)
- double **get\_sp\_energy** (int qnum) const
- double [H](#) (int n, double x) const  
*Method for calculating Hermite polynomials.*
- double [get\\_parameter](#) (int n)
- void [set\\_parameter](#) (double parameter, int n)

#### Protected Attributes

- double [w](#)  
*The oscillator frequency.*
- double \* [alpha](#)

- Pointer to the variational parameter  $\alpha$ . Shared address with all the *BasisFunction* subclasses.
- `double * k`  
 Pointer to  $\sqrt{\alpha*w}$ . Shared address with all the *BasisFunction* subclasses.
- `double * k2`  
 Pointer to  $\alpha*w$ . Shared address with all the *BasisFunction* subclasses.
- `double * exp_factor`  
 Pointer to a factor precalculated by `set_qnum_indie_terms()`. Shared address with all the *BasisFunction* subclasses.

## Friends

- class **ExpandedBasis**
- class **DiTransform**

### 3.1.1 Detailed Description

Harmonic Oscillator single particle wave function class. Uses the *HarmonicOscillator BasisFunction* subclasses auto-generated by SymPy through the *orbitalsGenerator* tool.

### 3.1.2 Member Function Documentation

**3.1.2.1** `double AlphaHarmonicOscillator::get_coulomb_element ( const arma::uvec & qnum_set )` [protected], [virtual]

For Quantum Dots, closed form expressions for the matrix elements exist.

Reimplemented from [Orbitals](#).

**3.1.2.2** `double AlphaHarmonicOscillator::get_dell_alpha_phi ( Walker * walker, int p, int q_num )` [protected], [virtual]

Overridden superclass method implementing closed form expressions using Hermite polynomials.

Reimplemented from [Orbitals](#).

**3.1.2.3** `double AlphaHarmonicOscillator::get_parameter ( int n )` [inline], [protected], [virtual]

#### Returns

The variational parameter  $\alpha$ .

Implements [Orbitals](#).

**3.1.2.4** `void AlphaHarmonicOscillator::get_qnums ( )` [protected]

Calculates the quantum numbers of the oscillator and stores them in the matrix *qnums*.

**3.1.2.5** `double AlphaHarmonicOscillator::H ( int n, double x ) const` [protected]

Method for calculating Hermite polynomials.

#### Parameters

$n$	The degree of the Hermite polynomial.
$x$	The argument for evaluating the polynomial.

**3.1.2.6** `void AlphaHarmonicOscillator::set_parameter ( double parameter, int n )` `[inline], [protected], [virtual]`

Sets a new value for the alpha and updates all the pointer values.

Implements [Orbitals](#).

**3.1.2.7** `void AlphaHarmonicOscillator::set_qnum_indie_terms ( Walker * walker, int i )` `[virtual]`

Calculates the exponential term shared by all oscillator function once pr. particle to save CPU-time.

See Also

[Orbitals::set\\_qnum\\_indie\\_terms\(\)](#)

Reimplemented from [Orbitals](#).

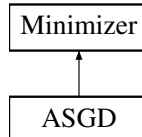
The documentation for this class was generated from the following files:

- `src/Orbitals/AlphaHarmonicOscillator/AlphaHarmonicOscillator.h`
- `src/Orbitals/AlphaHarmonicOscillator/AlphaHarmonicOscillator.cpp`

## 3.2 ASGD Class Reference

Implementation for the Adaptive Stochastic Gradient Descent method ([ASGD](#)) Used to find optimal variational parameters using adaptive step lengths.

Inheritance diagram for ASGD:



### Public Member Functions

- **ASGD** (`VMC *`, [MinimizerParams](#) &, const [ParParams](#) &, std::string path)
- void [minimize](#) ()

*Method for executing the minimization main solver.*

### Protected Member Functions

- void [get\\_total\\_grad](#) ()  
*Method for calculating the total gradient.*
- void [update\\_parameters](#) ()  
*Calculates the step and updates parameters.*
- void [output\\_cycle](#) ()  
*Standard output of the progress.*
- void [thermalize\\_walkers](#) ()  
*Thermalizes a set of walkers before the main loop.*
- double [f](#) (double x)  
*Function for calculating the adaptive step.*
- void [get\\_variational\\_gradients](#) ([Walker](#) \*walker, double e\_local)  
*Method for updating the vectors needed to calculate the total variational derivative.*

## Protected Attributes

- int `n_c`  
*The correlation length between storing two walkers after thermalization.*
- int `n_c_SGD`  
*The number of samples used to estimate expectation values.*
- int `SGDsamples`  
*The number of *ASGD* cycles.*
- int `n_walkers`  
*The number of walkers.*
- int `thermalization`  
*The number of thermalization cycles used on walkers.*
- int `sample`  
*The current *ASGD* cycle.*
- double `t_prev`  
*The previous *t*.*
- double `t`  
*The current *t*.*
- double `step`  
*The current calculates step.*
- double `max_step`  
*The maximum threshold on a step.*
- double `E`  
*The energy summation variable used to calculate the mean.*
- double `a`  
**ASGD* step parameter.*
- double `A`  
**ASGD* step parameter.*
- double `f_min`  
**ASGD* step parameter.*
- double `f_max`  
**ASGD* step parameter.*
- double `w`  
**ASGD* step parameter.*
- `Walker` \*\* `walkers`  
*The walkers used to sample expectation values.*
- `Walker` \*\* `trial_walkers`
- `arma::rowvec` `parameter`
- `arma::rowvec` `gradient`  
*Sumantion vector for the trial wave function's variational derivatives.*
- `arma::rowvec` `gradient_local`  
*Sumantion vector for the trial wave function's variational derivatives times the energy.*
- `arma::rowvec` `gradient_old`  
*The previous total gradient.*
- `arma::rowvec` `gradient_tot`  
*The current total gradient.*
- `stdoutASGD` \* `ASGDout`

## Friends

- class `stdoutASGD`

### 3.2.1 Detailed Description

Implementation for the Adaptive Stochastic Gradient Descent method ([ASGD](#)) Used to find optimal variational parameters using adaptive step lengths.

### 3.2.2 Member Function Documentation

#### 3.2.2.1 void ASGD::get\_total\_grad ( ) [protected]

Method for calculating the total gradient.

Updates the error estimator with statistics.

#### 3.2.2.2 void ASGD::get\_variational\_gradients ( Walker \* walker, double e\_local ) [protected]

Method for updating the vectors needed to calculate the total variational derivative.

Calculates the single particle variational derivatives V and accumulates V and V\*e\_local.

#### Parameters

<i>e_local</i>	The local energy of the current walker at the current time step.
----------------	--

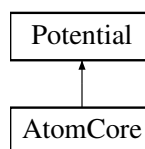
The documentation for this class was generated from the following files:

- src/Minimizer/ASGD/ASGD.h
- src/Minimizer/ASGD/ASGD.cpp

## 3.3 AtomCore Class Reference

Implementation of the Atom Core potential.  $-Z/r$ .

Inheritance diagram for AtomCore:



### Public Member Functions

- **AtomCore** ([GeneralParams](#) &)
- double [get\\_pot\\_E](#) (const [Walker](#) \*walker) const  
*Method for calculating a walker's potential energy.*

### Protected Attributes

- int [Z](#)  
*The core charge.*

## Additional Inherited Members

### 3.3.1 Detailed Description

Implementation of the Atom Core potential.  $-Z/r$ .

### 3.3.2 Member Function Documentation

#### 3.3.2.1 `double AtomCore::get_pot_E ( const Walker * walker ) const` [virtual]

Method for calculating a walker's potential energy.

Method overridden by subclasses.

#### Parameters

<i>walker</i>	The walker for which the potential energy should be calculated.
---------------	---

#### Returns

The potential energy.

Implements [Potential](#).

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

- `src/Potential/AtomCore/AtomCore.h`
- `src/Potential/AtomCore/AtomCore.cpp`

## 3.4 BasisFunctions Class Reference

The Superclass shell for orbital basis functions.

### Public Member Functions

- virtual double `eval` (const [Walker](#) \*walker, int i)=0  
*The method representing the orbitals functional expression.*

#### 3.4.1 Detailed Description

The Superclass shell for orbital basis functions.

Each single particle orbital has it's own implementation as a subclass of this class. A set of orbitals can then be loaded into the [Orbitals](#) `basis_function` vectors. An orbitalGenerator script is supplied to autogenerate CPP files using this class.

#### See Also

[Orbitals::basis\\_functions](#)  
[Orbitals::dell\\_basis\\_functions](#)  
[Orbitals::lapl\\_basis\\_functions](#)



### 3.4.2 Member Function Documentation

#### 3.4.2.1 virtual double BasisFunctions::eval ( const Walker \* walker, int i ) [pure virtual]

The method representing the orbitals functional expression.

##### Parameters

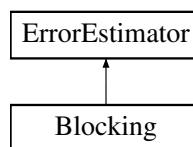
<i>walker</i>	The <a href="#">Walker</a> whose position the orbital is to be evaluated at.
<i>i</i>	The particle to be evaluated (Single particle function).

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

- src/BasisFunctions/BasisFunctions.h
- src/BasisFunctions/BasisFunctions.cpp

## 3.5 Blocking Class Reference

Inheritance diagram for Blocking:



### Public Member Functions

- **Blocking** (int [n\\_c](#), [ParParams](#) &pp, std::string filename="blocking\_out", std::string path="/", int n\_b=100, int maxb=10000, int minb=10, bool [rerun](#)=false)
- **Blocking** (int [n\\_c](#), std::string filename="blocking\_out", std::string path="/", int n\_b=100, int maxb=10000, int minb=10)
- double [estimate\\_error](#) ()  
*Estimates the error based on the subclass implementation.*
- void [get\\_initial\\_error](#) ()
- void [get\\_unique\\_blocks](#) (arma::Row< int > &block\_sizes, int &n)  
*Calculates the block sizes.*

### Protected Member Functions

- void [block\\_data](#) (int block\_size, double &var, double &mean)  
*Calculates the variance and mean of the dataset with the specified block size.*

### Protected Attributes

- arma::rowvec [local\\_block](#)
- int [min\\_block\\_size](#)  
*The minimum amount of samples in one block.*
- int [max\\_block\\_size](#)  
*The maximum amount of samples in one block.*
- int [n\\_block\\_samples](#)  
*The total amount of different block sizes.*

## Additional Inherited Members

### 3.5.1 Member Function Documentation

3.5.1.1 `void Blocking::block_data ( int block_size, double & var, double & mean )` `[protected]`

Calculates the variance and mean of the dataset with the specified block size.

#### Parameters

<i>block_size</i>	The number of samples in each block.
<i>var</i>	Reference to the variance of the block's means.
<i>mean</i>	Reference to the mean of the block's means. Needed to combine the variances from different processes.

3.5.1.2 `void Blocking::get_initial_error ( )`

Calculates the variance as in [SimpleVar](#)

3.5.1.3 `void Blocking::get_unique_blocks ( arma::Row< int > & block_sizes, int & n )`

Calculates the block sizes.

Due to integer division, alot of sizes becomes equal. Only unique block sizes are returned.

#### Parameters

<i>block_sizes</i>	Vector containing the block sizes
<i>n</i>	The number of unqiue block sizes

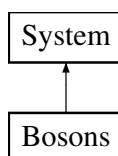
The documentation for this class was generated from the following files:

- `src/ErrorEstimator/Blocking/Blocking.h`
- `src/ErrorEstimator/Blocking/Blocking.cpp`

## 3.6 Bosons Class Reference

The Boson system class.

Inheritance diagram for Bosons:



### Public Member Functions

- **Bosons** ([GeneralParams](#) &, [Orbitals](#) \*)
- void [get\\_spatial\\_grad](#) ([Walker](#) \*walker, int particle) const  
*Method for calculating the changed part of the spatial gradient.*
- void [get\\_spatial\\_grad\\_full](#) ([Walker](#) \*walker) const

- Method for calculating the full spatial gradient.*
- double `get_spatial_ratio` (const [Walker](#) \*walker\_post, const [Walker](#) \*walker\_pre, int particle)
- Method for calculating the spatial wave function ratios between two subsequent time steps.*
- double `get_spatial_lapl_sum` ([Walker](#) \*walker) const
- Method for calculating the sum of all Laplacians for a given walker.*
- bool `allow_transition` ()
- Infinite potential to simulate bosonic behaviour.*
- void `copy_walker` (const [Walker](#) \*parent, [Walker](#) \*child) const
- void `update_walker` ([Walker](#) \*walker\_pre, const [Walker](#) \*walker\_post, int particle) const
- void `reset_walker` (const [Walker](#) \*walker\_pre, [Walker](#) \*walker\_post, int particle) const
- double `get_spatial_wf` (const [Walker](#) \*walker)
- void `initialize` ([Walker](#) \*walker)
- void `calc_for_newpos` (const [Walker](#) \*walker\_old, [Walker](#) \*walker\_new, int i)

## Protected Attributes

- int `a`
- The hard core radius for the infinite potential.*
- bool `overlap`
- True if the relative distance is less than the hard core radius.*

### 3.6.1 Detailed Description

The Boson system class.

### 3.6.2 Member Function Documentation

**3.6.2.1** void Bosons::calc\_for\_newpos ( const [Walker](#) \* walker\_old, [Walker](#) \* walker\_new, int i ) [inline],  
[virtual]

Does nothing.

Implements [System](#).

**3.6.2.2** void Bosons::copy\_walker ( const [Walker](#) \* parent, [Walker](#) \* child ) const [inline],[virtual]

Does nothing.

Implements [System](#).

**3.6.2.3** void Bosons::get\_spatial\_grad ( [Walker](#) \* walker, int particle ) const [virtual]

Method for calculating the changed part of the spatial gradient.

Depending on which particle we moved, one of the spatial wave function parts (it is split) will be unchanged.

Implements [System](#).

**3.6.2.4** double Bosons::get\_spatial\_wf ( const [Walker](#) \* walker ) [virtual]

The single particle states of each particle multiplied. Assumes the trial wave function initializes every particle in the same single particle state.

Implements [System](#).

**3.6.2.5** `void Bosons::initialize ( Walker * walker ) [inline],[virtual]`

Does nothing.

Implements [System](#).

**3.6.2.6** `void Bosons::reset_walker ( const Walker * walker_pre, Walker * walker_post, int particle ) const [inline],[virtual]`

Does nothing.

Implements [System](#).

**3.6.2.7** `void Bosons::update_walker ( Walker * walker_pre, const Walker * walker_post, int particle ) const [inline],[virtual]`

Does nothing.

Implements [System](#).

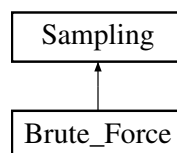
The documentation for this class was generated from the following files:

- src/System/Bosons/Bosons.h
- src/System/Bosons/Bosons.cpp

## 3.7 Brute\_Force Class Reference

Implementation of the Brute Force [QMC](#). Uses the Simple diffusion class. All methods are empty except for the energy necessities part which requires the gradients to be calculated (not using the Quantum Force).

Inheritance diagram for Brute\_Force:



### Public Member Functions

- **Brute\_Force** ([GeneralParams](#) &)
- void [update\\_walker](#) ([Walker](#) \*walker\_pre, const [Walker](#) \*walker\_post, int particle) const
- void [get\\_necessities](#) ([Walker](#) \*walker)  
*Method for calculating the sampling specific necessary values.*
- void [update\\_necessities](#) (const [Walker](#) \*walker\_pre, [Walker](#) \*walker\_post, int particle) const  
*Method for updating the sampling specific necessary values.*
- void [calculate\\_energy\\_necessities](#) ([Walker](#) \*walker) const  
*Method for calculating the sampling specific necessary values in order to compute the local energy.*
- void [copy\\_walker](#) (const [Walker](#) \*parent, [Walker](#) \*child) const  
*Method for copying the sampling specific parts of a walker.*
- void [reset\\_walker](#) (const [Walker](#) \*walker\_pre, [Walker](#) \*walker\_post, int particle) const

## Additional Inherited Members

### 3.7.1 Detailed Description

Implementation of the Brute Force [QMC](#). Uses the Simple diffusion class. All methods are empty except for the energy necessities part which requires the gradients to be calculated (not using the Quantum Force).

### 3.7.2 Member Function Documentation

**3.7.2.1** `void Brute_Force::copy_walker ( const Walker * parent, Walker * child ) const` `[inline], [virtual]`

Method for copying the sampling specific parts of a walker.

See Also

[QMC::copy\\_walker\(\)](#)

Implements [Sampling](#).

**3.7.2.2** `void Brute_Force::get_necessities ( Walker * walker )` `[inline], [virtual]`

Method for calculating the sampling specific necessary values.

Called after a trial position is set.

Implements [Sampling](#).

**3.7.2.3** `void Brute_Force::reset_walker ( const Walker * walker_pre, Walker * walker_post, int particle ) const` `[inline], [virtual]`

See Also

[QMC::reset\\_walker\(\)](#)

Implements [Sampling](#).

**3.7.2.4** `void Brute_Force::update_walker ( Walker * walker_pre, const Walker * walker_post, int particle ) const` `[inline], [virtual]`

See Also

[QMC::update\\_walker\(\)](#)

Implements [Sampling](#).

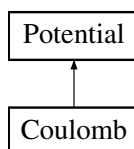
The documentation for this class was generated from the following files:

- `src/Sampling/Brute_Force/Brute_Force.h`
- `src/Sampling/Brute_Force/Brute_Force.cpp`

## 3.8 Coulomb Class Reference

Implementation of the [Coulomb](#) potential.  $1/r_{ij}$ .

Inheritance diagram for Coulomb:



## Public Member Functions

- **Coulomb** ([GeneralParams](#) &)
- double [get\\_pot\\_E](#) (const [Walker](#) \*walker) const  
*Method for calculating a walker's potential energy.*

## Additional Inherited Members

### 3.8.1 Detailed Description

Implementation of the [Coulomb](#) potential.  $1/r_{ij}$ .

### 3.8.2 Member Function Documentation

**3.8.2.1** double [Coulomb::get\\_pot\\_E](#) ( const [Walker](#) \* walker ) const [virtual]

Method for calculating a walker's potential energy.

Method overridden by subclasses.

#### Parameters

<i>walker</i>	The walker for which the potential energy should be calculated.
---------------	---

#### Returns

The potential energy.

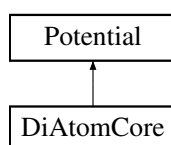
Implements [Potential](#).

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

- src/Potential/Coulomb/Coulomb.h
- src/Potential/Coulomb/Coulomb.cpp

## 3.9 DiAtomCore Class Reference

Inheritance diagram for DiAtomCore:



## Public Member Functions

- **DiAtomCore** ([GeneralParams](#) &gp)
- double [get\\_pot\\_E](#) (const [Walker](#) \*walker) const  
*Method for calculating a walker's potential energy.*

## Additional Inherited Members

### 3.9.1 Member Function Documentation

#### 3.9.1.1 double DiAtomCore::get\_pot\_E ( const Walker \* walker ) const [virtual]

Method for calculating a walker's potential energy.

Method overridden by subclasses.

#### Parameters

<i>walker</i>	The walker for which the potential energy should be calculated.
---------------	---

#### Returns

The potential energy.

Implements [Potential](#).

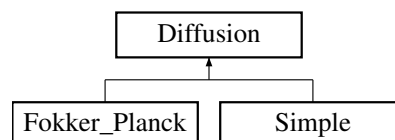
The documentation for this class was generated from the following files:

- src/Potential/DiAtomCore/DiAtomCore.h
- src/Potential/DiAtomCore/DiAtomCore.cpp

## 3.10 Diffusion Class Reference

Class containing rules for walker movement based on diffusion models. Serves as class member in the [Sampling](#) class. Brute force implies the [Simple](#) diffusion model, while [Importance Sampling](#) implies the Fokker Planck diffusion.

Inheritance diagram for Diffusion:



## Public Member Functions

- **Diffusion** (int n\_p, int dim, double [timestep](#), seed\_type [random\\_seed](#), double D)
- virtual double [get\\_new\\_pos](#) (const [Walker](#) \*walker, int i, int j)  
*Virtual function returning the new position.*
- virtual double [get\\_g\\_ratio](#) (const [Walker](#) \*walker\_post, const [Walker](#) \*walker\_pre) const =0  
*Calculates the [Diffusion](#) Green's function ratio needed by metropolis.*
- double [call\\_RNG](#) ()  
*Calls a uniform random number generator.*

- void **set\_qmc\_ptr** ([QMC](#) \*qmc)
- void **set\_dt** (double dt)  
*Function for altering the time step.*
- double **get\_dt** () const
- double **get\_std** () const

### Protected Attributes

- int **n\_p**
- int **dim**
- [QMC](#) \* qmc  
*The qmc main solver object. Not needed?*
- double **timestep**  
*The discrete time step.*
- double **D**  
*The diffusion constant.*
- long **random\_seed**  
*The random seed. Needs to be stored for some RNGs to work.*
- double **std**  
*The standard deviation from [QMC](#) stored for efficiency.  $\sqrt{2D \cdot \text{timestep}}$ .*

### 3.10.1 Detailed Description

Class containing rules for walker movement based on diffusion models. Serves as class member in the [Sampling](#) class. Brute force implies the [Simple](#) diffusion model, while [Importance Sampling](#) implies the Fokker Planck diffusion.

#### See Also

[Brute\\_Force](#), [Importance](#).

### 3.10.2 Member Function Documentation

#### 3.10.2.1 double Diffusion::call\_RNG ( )

Calls a uniform random number generator.

Returns a random uniform number on [0,1).

#### 3.10.2.2 virtual double Diffusion::get\_g\_ratio ( const Walker \* walker\_post, const Walker \* walker\_pre ) const [pure virtual]

Calculates the [Diffusion](#) Green's function ratio needed by metropolis.

#### Parameters

<i>walker_post</i>	<a href="#">Walker</a> at current time step.
<i>walker_pre</i>	<a href="#">Walker</a> at previous time step.

#### Returns

The [Diffusion](#) Green's function ratio.

Implemented in [Simple](#), and [Fokker\\_Planck](#).



3.10.2.3 `double Diffusion::get_new_pos ( const Walker * walker, int i, int j )` [virtual]

Virtual function returning the new position.

Returns the simple diffusion step if not overridden.

#### Parameters

<i>i</i>	Particle number.
<i>j</i>	dimension (x,y,z).

#### Returns

The new position (relative to the old).

Reimplemented in [Fokker\\_Planck](#), and [Simple](#).

3.10.2.4 `void Diffusion::set_dt ( double dt )`

Function for altering the time step.

Takes care of consequences. Time step should only be altered using this function.

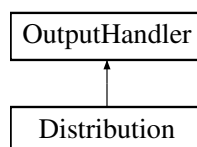
The documentation for this class was generated from the following files:

- `src/Diffusion/Diffusion.h`
- `src/Diffusion/Diffusion.cpp`

## 3.11 Distribution Class Reference

Class for calculating distribution functions such as the one-body density. Does not collect data itself, but works merely as a control organ for the [QMC](#) class, calling it's methods for storing position data.

Inheritance diagram for Distribution:



### Public Member Functions

- [Distribution](#) ([ParParams](#) &, std::string path, std::string name)  
*Constructor.*
- void [dump](#) ()  
*Methods for updating the output.*
- void [finalize](#) ()  
*Finalizes the output.*
- void [rerun](#) (int n\_p, int N, double bin\_edge=0)  
*Method for re-calculating the distribution given a stores set of position data.*

### Friends

- class [QMC](#)

## Additional Inherited Members

### 3.11.1 Detailed Description

Class for calculating distribution functions such as the one-body density. Does not collect data itself, but works merely as a control organ for the [QMC](#) class, calling it's methods for storing position data.

### 3.11.2 Constructor & Destructor Documentation

#### 3.11.2.1 `Distribution::Distribution ( ParParams & pp, std::string path, std::string name )`

Constructor.

##### Parameters

<i>path</i>	The path where output is stored (or read).
<i>name</i>	Name of the file.

### 3.11.3 Member Function Documentation

#### 3.11.3.1 `void Distribution::dump ( ) [inline],[virtual]`

Methods for updating the output.

Typically retrieves information through the solver pointers (given correct accessibility levels/friend)

Implements [OutputHandler](#).

#### 3.11.3.2 `void Distribution::finalize ( ) [inline],[virtual]`

Finalizes the output.

Closes file if use\_file flag is true. Can be overridden if more complex tasks needs to be done, such as calculating histograms etc.

##### See Also

[Distribution::finalize\(\)](#)

Reimplemented from [OutputHandler](#).

#### 3.11.3.3 `void Distribution::rerun ( int n_p, int N, double bin_edge = 0 )`

Method for re-calculating the distribution given a stores set of position data.

Scatters the data across nodes.

##### Parameters

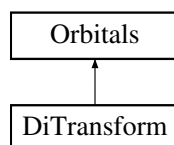
<i>n_p</i>	Number of particles in the set.
<i>N</i>	Number of mesh points used in the histogram.
<i>bin_edge</i>	The Cartesian position of the end points of the histogram.

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

- src/OutputHandler/Distribution/Distribution.h
- src/OutputHandler/Distribution/Distribution.cpp

## 3.12 DiTransform Class Reference

Inheritance diagram for DiTransform:



### Public Member Functions

- **DiTransform** ([GeneralParams](#) &gP, [VariationalParams](#) &vP)
- void [set\\_qnum\\_indie\\_terms](#) ([Walker](#) \*walker, int i)

### Protected Member Functions

- double [get\\_dell\\_alpha\\_phi](#) ([Walker](#) \*walker, int p, int q\_num)  
*! Sums contrib from nucleus1 and 2 in their mass center coordinates.*
- double [get\\_parameter](#) (int n)
- void [set\\_parameter](#) (double parameter, int n)
- double [phi](#) (const [Walker](#) \*walker, int particle, int q\_num)  
*Calculates the single particle wave function for a given walker's particle.*
- double [del\\_phi](#) (const [Walker](#) \*walker, int particle, int q\_num, int d)  
*Calculates the single particle wave function derivative for a given walker's particle and dimension.*
- double [lapl\\_phi](#) (const [Walker](#) \*walker, int particle, int q\_num)  
*Calculates the single particle wave function for a given walker's particle.*

### Static Protected Member Functions

- static double **minusPower** (int n)

### Protected Attributes

- double \* **R**
- [Orbitals](#) \* **nucleus1**
- [Orbitals](#) \* **nucleus2**
- [Walker](#) \* **walker\_nucleus1**
- [Walker](#) \* **walker\_nucleus2**

### 3.12.1 Member Function Documentation

**3.12.1.1** double [DiTransform::del\\_phi](#) ( const [Walker](#) \* *walker*, int *particle*, int *q\_num*, int *d* ) [protected],  
[virtual]

Calculates the single particle wave function derivative for a given walker's particle and dimension.

#### Parameters

<i>q_num</i>	The quantum number index.
<i>d</i>	The dimension for which the derivative should be calculated (x,y,z).

Reimplemented from [Orbitals](#).

**3.12.1.2** `double DiTransform::get_parameter ( int n )` `[inline]`, `[protected]`, `[virtual]`

Returns

The variational parameter alpha for all objects.

Implements [Orbitals](#).

**3.12.1.3** `double DiTransform::lapl_phi ( const Walker * walker, int particle, int q_num )` `[protected]`, `[virtual]`

Calculates the single particle wave function for a given walker's particle.

Parameters

<code><i>q_num</i></code>	The quantum number index.
---------------------------	---------------------------

Reimplemented from [Orbitals](#).

**3.12.1.4** `double DiTransform::phi ( const Walker * walker, int particle, int q_num )` `[protected]`, `[virtual]`

Calculates the single particle wave function for a given walker's particle.

Parameters

<code><i>q_num</i></code>	The quantum number index.
---------------------------	---------------------------

Reimplemented from [Orbitals](#).

**3.12.1.5** `void DiTransform::set_parameter ( double parameter, int n )` `[inline]`, `[protected]`, `[virtual]`

Calls methods in [hydrogenicOrbitals](#).

Implements [Orbitals](#).

**3.12.1.6** `void DiTransform::set_qnum_indie_terms ( Walker * walker, int i )` `[virtual]`

Calculates the exponential terms  $\exp(-r/n)$  for all needed *n* once pr. particle per core to save CPU-time.

See Also

[Orbitals::set\\_qnum\\_indie\\_terms\(\)](#)

Reimplemented from [Orbitals](#).

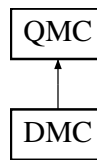
The documentation for this class was generated from the following files:

- `src/Orbitals/DiTransform/DiTransform.h`
- `src/Orbitals/DiTransform/DiTransform.cpp`

## 3.13 DMC Class Reference

Implementation of the [Diffusion](#) Monte-Carlo Method. Very little needs to be added when the [QMC](#) superclass holds all the general functionality.

Inheritance diagram for DMC:



### Public Member Functions

- **DMC** ([GeneralParams](#) &, [DMCparams](#) &, [SystemObjects](#) &, [ParParams](#) &, [VMC](#) \*vmc)  
*Constructor.*
- void [run\\_method](#) ()  
*Method used for executing the solver.*
- void [output](#) ()  
*Method for standard output.*

### Static Public Attributes

- static const int [K](#) = 50  
*Factor of empty space for walkers over initial walkers.*
- static const int **check\_thresh** = 25
- static const int **sendcount\_thresh** = 20

### Protected Member Functions

- void [set\\_trial\\_positions](#) ()  
*Method for setting the trial position of all [DMC](#) walkers.*
- void [iterate\\_walker](#) (int k)  
*Method for iterating a walker one time step.*
- void [Evolve\\_walker](#) (int k, double GB)  
*Method for the birth/death process of a walker.*
- void [bury\\_the\\_dead](#) ()  
*Method for cleaning up the dead walkers and compact the list.*
- void [update\\_energies](#) ()  
*Method for updating the [DMC](#) energy and calculating the new trial energy.*
- bool [move\\_authorized](#) (double A)
- void **reset\_parameters** ()
- void [node\\_comm](#) ()
- void [save\\_distribution](#) ()  
*Method for storing positional data for the Distribution.*
- void [switch\\_souls](#) (int root, int root\_id, int dest, int dest\_id)  
*Method for sending a walker between two nodes.*
- void [normalize\\_population](#) ()  
*Method for evening out the number of walkers on each node.*
- void [free\\_walkers](#) ()  
*Method which deletes all walkers.*

## Protected Attributes

- bool [thermalized](#)  
*Flag used to indicate whether to start sampling or not.*
- int [n\\_w\\_last](#)  
*The amount of walkers at the time the walker loop was initiated.*
- int [n\\_w\\_tot](#)  
*The total number of walkers across all nodes.*
- arma::uvec [n\\_w\\_list](#)  
*List of the number of walkers of each node.*
- bool [force\\_comm](#)  
*Flag set true if population should be renormalized.*
- int **deaths**
- int **block\_size**
- int **samples**
- double [dmc\\_E](#)  
*The [DMC](#) energy.*
- double [dmc\\_E\\_unscaled](#)  
*The accumulative [DMC](#) energy: The sum of all previous trial energies.*
- double [E\\_T](#)  
*The trial energy at the current cycle.*
- double [E](#)  
*The accumulative energy for each cycle.*
- [stdoutDMC](#) \* **DMCout**

## Friends

- class **stdoutDMC**

### 3.13.1 Detailed Description

Implementation of the [Diffusion](#) Monte-Carlo Method. Very little needs to be added when the [QMC](#) superclass holds all the general functionality.

### 3.13.2 Member Function Documentation

#### 3.13.2.1 void DMC::Evolve\_walker ( int *k*, double *GB* ) [protected]

Method for the birth/death process of a walker.

##### Parameters

<i>GB</i>	The branching Green's Function.
<i>k</i>	The index of the walker.

#### 3.13.2.2 void DMC::iterate\_walker ( int *k* ) [protected]

Method for iterating a walker one time step.

## Parameters

<i>thermalized</i>	Flag to indicate whether to start sampling or not.
<i>k</i>	The index of the walker.

3.13.2.3 `bool DMC::move_authorized ( double A )` `[protected]`, `[virtual]`

In case of [DMC](#), we must let the system have the possibility to override the metropolis test (fixed node approximation in case of a Fermion system)

Implements [QMC](#).

3.13.2.4 `void DMC::node_comm ( )` `[protected]`, `[virtual]`

For each process: -Calculates the total number of walkers. -Sums up the energies sampled. -Sums up the total number of samples made.

Implements [QMC](#).

3.13.2.5 `void DMC::save_distribution ( )` `[protected]`, `[virtual]`

Method for storing positional data for the Distribtuon.

Stores the position data from all currently alive [DMC](#) walkers every `dist_tresh` cycle.

Implements [QMC](#).

3.13.2.6 `void DMC::set_trial_positions ( )` `[protected]`, `[virtual]`

Method for setting the trial position of all [DMC](#) walkers.

In case [VMC](#) is not run prior to [DMC](#), trial positions must be set.

Implements [QMC](#).

3.13.2.7 `void DMC::switch_souls ( int root, int root_id, int dest, int dest_id )` `[protected]`

Method for sending a walker between two nodes.

## Parameters

<i>root</i>	The node from which the walker is sent.
<i>root_id</i>	The index of the walker being sent from root.
<i>dest</i>	The node which receives the walker.
<i>dest_id</i>	The index where the walker is to be received.

## See Also

[Walker::send\\_soul\(\)](#), [Walker::recv\\_soul\(\)](#)

## 3.13.3 Member Data Documentation

3.13.3.1 `const int DMC::K = 50` `[static]`

Factor of empty space for walkers over initial walkers.

## See Also

QMC::QMC()

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

- src/QMC/DMC/DMC.h
- src/QMC/DMC/DMC.cpp

### 3.14 DMCparams Struct Reference

Struct used to initialize [DMC](#) parameters.

#### Public Attributes

- int [n\\_c](#)  
*The number of cycles.*
- int [therm](#)  
*Thermalization cycles.*
- int [n\\_b](#)  
*Number of block samples pr. walker pr. cycle.*
- int [n\\_w](#)  
*Number of walkers.*
- double [dt](#)  
*Time step.*

#### 3.14.1 Detailed Description

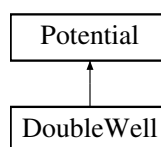
Struct used to initialize [DMC](#) parameters.

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

- src/structs.h

### 3.15 DoubleWell Class Reference

Inheritance diagram for DoubleWell:



#### Public Member Functions

- **DoubleWell** ([GeneralParams](#) &gp)
- double [get\\_pot\\_E](#) (const [Walker](#) \*walker) const  
*Method for calculating a walker's potential energy.*



## Additional Inherited Members

### 3.15.1 Member Function Documentation

3.15.1.1 `double DoubleWell::get_pot.E ( const Walker * walker ) const` [virtual]

Method for calculating a walker's potential energy.

Method overridden by subclasses.

#### Parameters

<i>walker</i>	The walker for which the potential energy should be calculated.
---------------	---

#### Returns

The potential energy.

Implements [Potential](#).

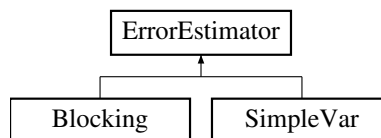
The documentation for this class was generated from the following files:

- src/Potential/DoubleWell/DoubleWell.h
- src/Potential/DoubleWell/DoubleWell.cpp

## 3.16 ErrorEstimator Class Reference

Class handling error estimations of the [QMC](#) methods. The [QMC](#) class holds an object of this type, calling the `update_data` function in order to update the sampling pool. `finalize()` then either dumps the samples to file for later processing, or calculates an estimate.

Inheritance diagram for ErrorEstimator:



### Public Member Functions

- [ErrorEstimator](#) (int [n\\_c](#), std::string filename, std::string path, bool parallel, int node, int n\_nodes, bool [re-run](#)=false)  
*Constructor.*
- double [combine\\_variance](#) (double var, double mean=0, int n=0)  
*Calculates the combined variance of n\_nodes variances.*
- void [finalize](#) ()
- void [node\\_comm\\_gather\\_data](#) ()
- void [node\\_comm\\_scatter\\_data](#) ()
- void [init\\_file](#) ()  
*Opens a file with filename at path supplied in constructor.*
- virtual double [estimate\\_error](#) ()=0  
*Estimates the error based on the subclass implementation.*
- virtual void [update\\_data](#) (double val)  
*Adds values to the data vector.*

## Static Public Member Functions

- static double `combine_mean` (double mean, int n, int n\_tot=0)  
*Calculates the combined mean of `n_nodes` means.*

## Public Attributes

- bool `data_to_file`  
*If true, the data vector are stored to file.*
- bool `output_to_file`  
*If `init_file()` method is called, this flag is true.*

## Protected Attributes

- int `n_c`  
*Size of the data vector.*
- int `i`  
*Count variable for the data vector.*
- bool `parallel`
- bool `is_master`
- int `node`
- int `n_nodes`
- bool `rerun`  
*If false, data is assumed to already exist.*
- std::string `filename`
- std::string `path`
- std::ofstream `file`
- arma::rowvec `data`  
*The vector containing the samples used in error calculation.*

### 3.16.1 Detailed Description

Class handling error estimations of the `QMC` methods. The `QMC` class holds an object of this type, calling the `update_data` function in order to update the sampling pool. `finalize()` then either dumps the samples to file for later processing, or calculates an estimate.

### 3.16.2 Constructor & Destructor Documentation

- 3.16.2.1 `ErrorEstimator::ErrorEstimator ( int n_c, std::string filename, std::string path, bool parallel, int node, int n_nodes, bool rerun = false )`

Constructor.

#### Parameters

<code>n_c</code>	The expected number of samples to be stored.
<code>filename</code>	The name of the file. Only necessary if <code>init_file()</code> is called.
<code>path</code>	The path where the data and/or the file is stored (or read).

### 3.16.3 Member Function Documentation

**3.16.3.1** `double ErrorEstimator::combine_mean ( double mean, int n, int n_tot = 0 ) [static]`

Calculates the combined mean of `n_nodes` means.

Only useful for parallel calls.

**Parameters**

<i>mean</i>	The local mean on an individual node
<i>n</i>	The number of samples used to calculate the local mean.
<i>n_tot</i>	The total number of samples on all nodes. Calculated if not supplied.

**3.16.3.2** `double ErrorEstimator::combine_variance ( double var, double mean = 0, int n = 0 )`

Calculates the combined variance of `n_nodes` variances.

Only useful for parallel calls.

**Parameters**

<i>var</i>	The local variance on an individual node
<i>mean</i>	The local mean on an individual node
<i>n_tot</i>	The total number of samples used on all nodes.

**3.16.3.3** `void ErrorEstimator::finalize ( )`

if `[output_to_file]`: Closes opened files if `[data_to_file]`: Stores accumulated data. if data vector was used, it's memory is freed.

**3.16.3.4** `void ErrorEstimator::init_file ( )`

Opens a file with filename at path supplied in constructor.

Subclass implementations can call this function. Superclass does not.

**3.16.3.5** `void ErrorEstimator::node_comm_gather_data ( )`

Gathers the data vectors from all processes into a single one on the master node.

**3.16.3.6** `void ErrorEstimator::node_comm_scatter_data ( )`

Exact reverse of [node\\_comm\\_gather\\_data\(\)](#)

**3.16.3.7** `void ErrorEstimator::update_data ( double val ) [virtual]`

Adds values to the data vector.

Can be overridden if storage is not wanted.

**Parameters**

<i>val</i>	A local sample of the quantity of which the error is calculated
------------	---

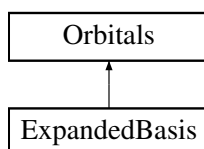
Reimplemented in [SimpleVar](#).

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

- src/ErrorEstimator/ErrorEstimator.h
- src/ErrorEstimator/ErrorEstimator.cpp

### 3.17 ExpandedBasis Class Reference

Inheritance diagram for ExpandedBasis:



#### Public Member Functions

- **ExpandedBasis** ([GeneralParams](#) &gp, [Orbitals](#) \*basis, int m, std::string coeffPath)
- double [phi](#) (const [Walker](#) \*walker, int particle, int q\_num)  
*Calculates the single particle wave function for a given walker's particle.*
- double [del\\_phi](#) (const [Walker](#) \*walker, int particle, int q\_num, int d)  
*Calculates the single particle wave function derivative for a given walker's particle and dimension.*
- double [lapl\\_phi](#) (const [Walker](#) \*walker, int particle, int q\_num)  
*Calculates the single particle wave function for a given walker's particle.*
- void [set\\_qnum\\_indie\\_terms](#) ([Walker](#) \*walker, int i)  
*Calculates single particle wave function terms which are independent of the quantum numbers.*

#### Protected Attributes

- int **basis\_size**
- arma::mat **coeffs**
- [Orbitals](#) \* **basis**

#### Additional Inherited Members

##### 3.17.1 Member Function Documentation

3.17.1.1 double ExpandedBasis::del\_phi ( const Walker \* walker, int particle, int q\_num, int d ) [virtual]

Calculates the single particle wave function derivative for a given walker's particle and dimension.

##### Parameters

<i>q_num</i>	The quantum number index.
<i>d</i>	The dimension for which the derivative should be calculated (x,y,z).

Reimplemented from [Orbitals](#).

3.17.1.2 double ExpandedBasis::lapl\_phi ( const Walker \* walker, int particle, int q\_num ) [virtual]

Calculates the single particle wave function for a given walker's particle.

## Parameters

<code>q_num</code>	The quantum number index.
--------------------	---------------------------

Reimplemented from [Orbitals](#).

3.17.1.3 `double ExpandedBasis::phi ( const Walker * walker, int particle, int q_num )` [virtual]

Calculates the single particle wave function for a given walker's particle.

## Parameters

<code>q_num</code>	The quantum number index.
--------------------	---------------------------

Reimplemented from [Orbitals](#).

3.17.1.4 `void ExpandedBasis::set_qnum_indep_terms ( Walker * walker, int i )` [inline],[virtual]

Calculates single particle wave function terms which are independent of the quantum numbers.

If a term in the single particle functions are independent of the quantum number, this function can be overridden to calculate them beforehand (for each particle), and rather extract the value instead of recalculating.

## Parameters

<code>i</code>	Particle number.
----------------	------------------

Reimplemented from [Orbitals](#).

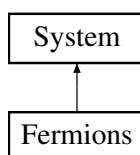
The documentation for this class was generated from the following files:

- `src/Orbitals/ExpandedBasis/ExpandedBasis.h`
- `src/Orbitals/ExpandedBasis/ExpandedBasis.cpp`

## 3.18 Fermions Class Reference

The Fermion system class.

Inheritance diagram for Fermions:



### Public Member Functions

- **Fermions** ([GeneralParams](#) &, [Orbitals](#) \*)
- void [get\\_spatial\\_grad](#) ([Walker](#) \*walker, int particle) const  
*Method for calculating the changed part of the spatial gradient.*
- void [get\\_spatial\\_grad\\_full](#) ([Walker](#) \*walker) const  
*Method for calculating the full spatial gradient.*
- double [get\\_spatial\\_ratio](#) (const [Walker](#) \*walker\_post, const [Walker](#) \*walker\_pre, int particle)  
*Method for calculating the spatial wave function ratios between two subsequent time steps.*

- double `get_spatial_lapl_sum` (`Walker *walker`) const  
*Method for calculating the sum of all Laplacians for a given walker.*
- bool `allow_transition` ()  
*Fixed node approximation.*
- void `copy_walker` (const `Walker *parent`, `Walker *child`) const
- void `update_walker` (`Walker *walker_pre`, const `Walker *walker_post`, int particle) const
- void `reset_walker` (const `Walker *walker_pre`, `Walker *walker_post`, int particle) const
- double `get_spatial_wf` (const `Walker *walker`)
- void `initialize` (`Walker *walker`)
- void `calc_for_newpos` (const `Walker *walker_old`, `Walker *walker_new`, int i)

## Protected Member Functions

- void `make_merged_inv` (`Walker *walker`)  
*Method for calculating the Slater matrix inverse.*
- void `update_inverse` (const `Walker *walker_old`, `Walker *walker_new`, int particle)  
*Method for updating the inverse given that we moved one particle.*

## Protected Attributes

- arma::rowvec `l`  
*The diagonal of the new slater matrix times the old slater inverse.*
- bool `node_crossed`  
*True if the spatial ratio is negative.*

### 3.18.1 Detailed Description

The Fermion system class.

### 3.18.2 Member Function Documentation

**3.18.2.1** void `Fermions::calc_for_newpos` ( const `Walker * walker_old`, `Walker * walker_new`, int `i` ) [inline],  
[virtual]

When a particle is moved, the inverse is updated.

Implements [System](#).

**3.18.2.2** void `Fermions::copy_walker` ( const `Walker * parent`, `Walker * child` ) const [inline], [virtual]

Copies the inverse.

Implements [System](#).

**3.18.2.3** void `Fermions::get_spatial_grad` ( `Walker * walker`, int `particle` ) const [virtual]

Method for calculating the changed part of the spatial gradient.

Depending on which particle we moved, one of the spatial wave function parts (it is split) will be unchanged.

Implements [System](#).

**3.18.2.4** `double Fermions::get_spatial_wf ( const Walker * walker )` `[inline],[virtual]`

The determinant of each spin value multiplied.

Implements [System](#).

**3.18.2.5** `void Fermions::initialize ( Walker * walker )` `[inline],[virtual]`

Calculates the inverse.

Implements [System](#).

**3.18.2.6** `void Fermions::make_merged_inv ( Walker * walker )` `[protected]`

Method for calculating the Slater matrix inverse.

The merged inverse is made by concatenating the two slater matrix inverses. This way we can sum freely over particles without having to if-test on the spin.

**3.18.2.7** `void Fermions::reset_walker ( const Walker * walker_pre, Walker * walker_post, int particle ) const` `[inline],[virtual]`

Resets the inverse.

Implements [System](#).

**3.18.2.8** `void Fermions::update_walker ( Walker * walker_pre, const Walker * walker_post, int particle ) const` `[inline],[virtual]`

Updates the inverse.

Implements [System](#).

### 3.18.3 Member Data Documentation

**3.18.3.1** `arma::rowvec Fermions::I` `[protected]`

The diagonal of the new slater matrix times the old slater inverse.

Needed for updating the inverse. Stored because only half of the vector is changed when moving one particle.

See Also

[System::set\\_spin\\_state\(\)](#)

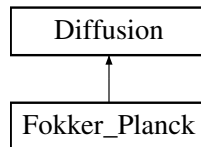
The documentation for this class was generated from the following files:

- `src/System/Fermions/Fermions.h`
- `src/System/Fermions/Fermions.cpp`

## 3.19 Fokker\_Planck Class Reference

Anisotropic diffusion by the Fokker-Planck equation.

Inheritance diagram for Fokker\_Planck:



## Public Member Functions

- **Fokker\_Planck** (int n\_p, int dim, double timestep, seed\_type random\_seed, double D=0.5)
- double **get\_g\_ratio** (const Walker \*walker\_post, const Walker \*walker\_pre) const  
*Calculates the Diffusion Green's function ratio needed by metropolis.*
- double **get\_new\_pos** (const Walker \*walker, int i, int j)  
*Virtual function returning the new position.*

## Additional Inherited Members

### 3.19.1 Detailed Description

Anisotropic diffusion by the Fokker-Planck equation.

### 3.19.2 Member Function Documentation

**3.19.2.1** double Fokker\_Planck::get\_g\_ratio ( const Walker \* walker\_post, const Walker \* walker\_pre ) const  
[virtual]

Calculates the Diffusion Green's function ratio needed by metropolis.

#### Parameters

<i>walker_post</i>	Walker at current time step.
<i>walker_pre</i>	Walker at previous time step.

#### Returns

The Diffusion Green's function ratio.

Implements Diffusion.

**3.19.2.2** double Fokker\_Planck::get\_new\_pos ( const Walker \* walker, int i, int j ) [inline], [virtual]

Virtual function returning the new position.

Returns the simple diffusion step if not overridden.

#### Parameters

<i>i</i>	Particle number.
<i>j</i>	dimension (x,y,z).

#### Returns

The new position (relative to the old).

Reimplemented from Diffusion.



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

- src/Diffusion/Fokker\_Planck/Fokker\_Planck.h
- src/Diffusion/Fokker\_Planck/Fokker\_Planck.cpp

## 3.20 GeneralParams Struct Reference

Struct used to initialize general parameters.

### Public Attributes

- int `n_p`  
*The number of particles.*
- int `dim`  
*The dimension.*
- seed\_type `random_seed`  
*The random number generator's seed.*
- double `systemConstant`  
*The constant used in systems.*
- double `R`
- bool `deadlock`  
*Center of mass coordinate for diatomic systems.*
- double `deadlock_x`  
*If true, freezes one particle;.*
- bool `doMIN`  
*Position of the locked particle. y=z=0;.*
- bool `doVMC`
- bool `doDMC`
- bool `do_blocking`
- bool `use_jastrow`
- bool `use_coulomb`
- std::string `system`  
*String specifying the system type, e.g. "Atoms".*
- std::string `sampling`  
*String specifying the sampling type, e.g. "IS".*
- std::string `runpath`  
*The directory which the simulation is set to run.*

### 3.20.1 Detailed Description

Struct used to initialize general parameters.

### 3.20.2 Member Data Documentation

#### 3.20.2.1 double GeneralParams::systemConstant

The constant used in systems.

e.g. charge for atoms and oscillator frequency for quantum dots.

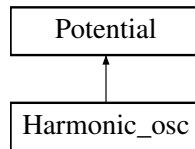
The documentation for this struct was generated from the following file:

- src/structs.h

### 3.21 Harmonic\_osc Class Reference

Implementation of the Harmonic Oscillator potential.  $0.5*w**2*r**2$ .

Inheritance diagram for Harmonic\_osc:



#### Public Member Functions

- **Harmonic\_osc** ([GeneralParams](#) &)
- double [get\\_pot\\_E](#) (const [Walker](#) \*walker) const  
*Method for calculating a walker's potential energy.*

#### Protected Attributes

- double [w](#)  
*The oscillator frequency.*

#### Additional Inherited Members

##### 3.21.1 Detailed Description

Implementation of the Harmonic Oscillator potential.  $0.5*w**2*r**2$ .

##### 3.21.2 Member Function Documentation

**3.21.2.1** double `Harmonic_osc::get_pot_E ( const Walker * walker ) const` `[virtual]`

Method for calculating a walker's potential energy.

Method overridden by subclasses.

#### Parameters

<i>walker</i>	The walker for which the potential energy should be calculated.
---------------	---

#### Returns

The potential energy.

Implements [Potential](#).

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

- `src/Potential/Harmonic_osc/Harmonic_osc.h`
- `src/Potential/Harmonic_osc/Harmonic_osc.cpp`

### 3.22 HartreeFock Class Reference

## Public Member Functions

- **HartreeFock** (int m, [Orbitals](#) \*sp\_basis, double thresh=1e-5)
- void **run\_method** ()

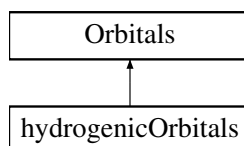
The documentation for this class was generated from the following files:

- src/HartreeFock/HartreeFock.h
- src/HartreeFock/HartreeFock.cpp

## 3.23 hydrogenicOrbitals Class Reference

Hydrogen-like single particle wave function class. Uses the hydrogenic BasisFunction subclasses auto-generated by SymPy through the orbitalsGenerator tool.

Inheritance diagram for hydrogenicOrbitals:



## Public Member Functions

- **hydrogenicOrbitals** ([GeneralParams](#) &, [VariationalParams](#) &)
- void **set\_qnum\_indie\_terms** ([Walker](#) \*walker, int i)

## Protected Member Functions

- double **get\_dell\_alpha\_phi** ([Walker](#) \*walker, int p, int qnum)  
*Method for calculating a single particle wave functions variational derivative.*
- double **get\_sp\_energy** (int qnum) const
- double **get\_coulomb\_element** (const arma::uvec &qnum\_set)  
*Method for calculating the anti-symmetrized Coulomb matrix elements.*
- void **get\_qnums** ()
- double **get\_parameter** (int n)
- void **set\_parameter** (double parameter, int n)

## Protected Attributes

- int **Z**  
*The charge of the core.*
- double \* **alpha**  
*Pointer to the variational parameter alpha. Shared address with all the BasisFunction subclasses.*
- double \* **k**  
*Pointer to alpha\*Z. Shared address with all the BasisFunction subclasses.*
- double \* **k2**  
*Pointer (alpha\*Z)^2. Shared address with all the BasisFunction subclasses.*
- double \* **exp\_factor\_n1**  
*Pointer to a factor precalculated by set\_qnum\_indie\_terms(). Shared address with all the BasisFunction subclasses.*

- double \* [exp\\_factor\\_n2](#)  
Pointer to a factor precalculated by [set\\_qnum\\_indie\\_terms\(\)](#). Shared address with all the BasisFunction subclasses.
- double \* [exp\\_factor\\_n3](#)  
Pointer to a factor precalculated by [set\\_qnum\\_indie\\_terms\(\)](#). Shared address with all the BasisFunction subclasses.
- double \* [exp\\_factor\\_n4](#)  
Pointer to a factor precalculated by [set\\_qnum\\_indie\\_terms\(\)](#). Shared address with all the BasisFunction subclasses.

## Friends

- class **ExpandedBasis**
- class **DiTransform**

### 3.23.1 Detailed Description

Hydrogen-like single particle wave function class. Uses the hydrogenic BasisFunction subclasses auto-generated by SymPy through the orbitalsGenerator tool.

### 3.23.2 Member Function Documentation

**3.23.2.1** `double hydrogenicOrbitals::get_coulomb_element ( const arma::uvec & qnum_set )` `[protected]`,  
`[virtual]`

Method for calculating the anti-symmetrized Coulomb matrix elements.

Used by Hartree Fock

Reimplemented from [Orbitals](#).

**3.23.2.2** `double hydrogenicOrbitals::get_dell_alpha_phi ( Walker * walker, int p, int qnum )` `[protected]`,  
`[virtual]`

Method for calculating a single particle wave functions variational derivative.

#### Parameters

<i>i</i>	The particle number.
----------	----------------------

Reimplemented from [Orbitals](#).

**3.23.2.3** `double hydrogenicOrbitals::get_parameter ( int n )` `[inline]`, `[protected]`, `[virtual]`

#### Returns

The variational parameter alpha.

Implements [Orbitals](#).

**3.23.2.4** `void hydrogenicOrbitals::set_parameter ( double parameter, int n )` `[inline]`, `[protected]`,  
`[virtual]`

Sets a new value for the alpha and updates all the pointer values.

Implements [Orbitals](#).

3.23.2.5 `void hydrogenicOrbitals::set_qnum_indie_terms ( Walker * walker, int i ) [virtual]`

Calculates the exponential terms  $\exp(-r/n)$  for all needed  $n$  once pr. particle to save CPU-time.

See Also

[Orbitals::set\\_qnum\\_indie\\_terms\(\)](#)

Reimplemented from [Orbitals](#).

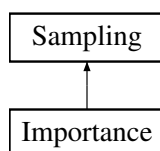
The documentation for this class was generated from the following files:

- `src/Orbitals/hydrogenicOrbitals/hydrogenicOrbitals.h`
- `src/Orbitals/hydrogenicOrbitals/hydrogenicOrbitals.cpp`

## 3.24 Importance Class Reference

Implementation of [Importance](#) sampled [QMC](#). Using the Fokker-Planck diffusion class. Introduces the Quantum Force.

Inheritance diagram for Importance:



### Public Member Functions

- **Importance** ([GeneralParams](#) &)
- `void update_walker (Walker *walker_pre, const Walker *walker_post, int particle) const`
- `void reset_walker (const Walker *walker_pre, Walker *walker_post, int particle) const`
- `void get_necessities (Walker *walker)`
- `void update_necessities (const Walker *walker_pre, Walker *walker_post, int particle) const`
- `void calculate_energy_necessities (Walker *walker) const`
- `void copy_walker (const Walker *parent, Walker *child) const`

### Additional Inherited Members

#### 3.24.1 Detailed Description

Implementation of [Importance](#) sampled [QMC](#). Using the Fokker-Planck diffusion class. Introduces the Quantum Force.

#### 3.24.2 Member Function Documentation

3.24.2.1 `void Importance::calculate_energy_necessities ( Walker * walker ) const [inline],[virtual]`

No energy necessities (they are already calculated).

Implements [Sampling](#).

3.24.2.2 `void Importance::copy_walker ( const Walker * parent, Walker * child ) const` `[virtual]`

The gradients and the Quantum force is copied.

Implements [Sampling](#).

3.24.2.3 `void Importance::get_necessities ( Walker * walker )` `[inline],[virtual]`

The gradients and the Quantum force are calculated.

Implements [Sampling](#).

3.24.2.4 `void Importance::reset_walker ( const Walker * walker_pre, Walker * walker_post, int particle ) const`  
`[virtual]`

The parts of the gradients with the same spin as the moved particle are reset.

Implements [Sampling](#).

3.24.2.5 `void Importance::update_necessities ( const Walker * walker_pre, Walker * walker_post, int particle ) const`  
`[inline],[virtual]`

The gradients are updated and the Quantum force is re-calculated.

Implements [Sampling](#).

3.24.2.6 `void Importance::update_walker ( Walker * walker_pre, const Walker * walker_post, int particle ) const`  
`[virtual]`

The parts of the gradients with the same spin as the moved particle are updated.

Implements [Sampling](#).

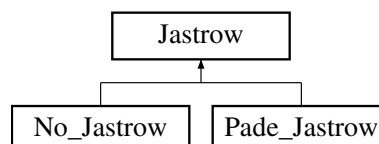
The documentation for this class was generated from the following files:

- `src/Sampling/Importance/Importance.h`
- `src/Sampling/Importance/Importance.cpp`

## 3.25 Jastrow Class Reference

The class representing the [Jastrow](#) correlation functions Holds all data concerning the [Jastrow](#) function and it's influence on the [QMC](#) algorithm.

Inheritance diagram for Jastrow:



### Public Member Functions

- **Jastrow** (int *n\_p*, int *dim*)
- virtual void [initialize](#) ()=0

- virtual double `get_val` (const `Walker` \*walker) const =0
- virtual double `get_j_ratio` (const `Walker` \*walker\_new, const `Walker` \*walker\_old, int i) const =0  
*Calculates the ratio of the `Jastrow` factor needed by metropolis.*
- virtual void `get_grad` (`Walker` \*walker) const =0
- virtual void `get_grad` (const `Walker` \*walker\_pre, `Walker` \*walker\_post, int i) const =0  
*Updates the gradient for a new particle move.*
- virtual void `get_dj_matrix` (`Walker` \*walker, int i) const =0  
*Updates the summation factors of `Jastrow` factors closed form expressions.*
- void `get_dj_matrix` (`Walker` \*walker) const  
*Calculates the summation factors of `Jastrow` factors closed form expressions.*
- virtual double `get_lapl_sum` (`Walker` \*walker) const =0  
*Method for calculating the Laplacian.*

### Protected Member Functions

- virtual double `get_parameter` (int n)=0  
*Returns variational parameters.*
- virtual void `set_parameter` (double param, int n)=0  
*Sets variational parameters.*
- virtual double `get_variational_derivative` (const `Walker` \*walker, int n)  
*Calculates the derivative of the `Jastrow` factor with respect to a variational parameter.*
- double `get_derivative_num` (`Walker` \*walker, int i, int d) const  
*Numerical Cartesian derivative.*
- double `get_laplaciansum_num` (`Walker` \*walker) const  
*Numerical Cartesian Laplacian.*

### Protected Attributes

- int `n_p`
- int `n2`
- int `dim`
- bool `active`  
*Parameter false if `No_Jastrow` is loaded.*

### Friends

- class `Minimizer`
- class `ASGD`
- class `stdoutASGD`

#### 3.25.1 Detailed Description

The class representing the `Jastrow` correlation functions Holds all data concerning the `Jastrow` function and it's influence on the `QMC` algorithm.

### 3.25.2 Member Function Documentation

3.25.2.1 `double Jastrow::get_derivative_num ( Walker * walker, int i, int d ) const` [protected]

Numerical Cartesian derivative.

For use in [get\\_grad\(\)](#) when no closed form expression is implemented.

#### Parameters

<i>i</i>	Particle number.
<i>d</i>	Dimension (x,y,z).

3.25.2.2 `virtual void Jastrow::get_dJ_matrix ( Walker * walker, int i ) const` [pure virtual]

Updates the summation factors of [Jastrow](#) factors closed form expressions.

Used to optimize the calculations as few of these terms change as we move a particle.

#### Parameters

<i>i</i>	Particle number.
----------	------------------

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

3.25.2.3 `void Jastrow::get_dJ_matrix ( Walker * walker ) const`

Calculates the summation factors of [Jastrow](#) factors closed form expressions.

Used to optimize the calculations as few of these terms change as we move a particle.

3.25.2.4 `virtual void Jastrow::get_grad ( Walker * walker ) const` [pure virtual]

Calculates the entire Cartesian gradient.

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

3.25.2.5 `virtual void Jastrow::get_grad ( const Walker * walker_pre, Walker * walker_post, int i ) const` [pure virtual]

Updates the gradient for a new particle move.

#### Parameters

<i>walker_post</i>	<a href="#">Walker</a> at current time step
<i>walker_pre</i>	<a href="#">Walker</a> at previous time step
<i>i</i>	Particle number.

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

3.25.2.6 `virtual double Jastrow::get_j_ratio ( const Walker * walker_new, const Walker * walker_old, int i ) const` [pure virtual]

Calculates the ratio of the [Jastrow](#) factor needed by metropolis.



## Parameters

<i>walker_new</i>	<a href="#">Walker</a> at current time step
<i>walker_old</i>	<a href="#">Walker</a> at previous time step
<i>i</i>	The particle number.

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

**3.25.2.7** `virtual double Jastrow::get_lapl_sum ( Walker * walker ) const` [pure virtual]

Method for calculating the Laplacian.

Calculates the sum of all particles Laplacians.

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

**3.25.2.8** `double Jastrow::get_laplaciansum_num ( Walker * walker ) const` [protected]

Numerical Cartesian Laplacian.

For use in [get\\_lapl\\_sum\(\)](#) when no closed form expression is implemented.

**3.25.2.9** `virtual double Jastrow::get_parameter ( int n )` [protected], [pure virtual]

Returns variational parameters.

## Parameters

<i>n</i>	The index of the sought variational parameter
----------	---

## Returns

Variational parameter with index [n]

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

**3.25.2.10** `virtual double Jastrow::get_val ( const Walker * walker ) const` [pure virtual]

Calculates the value of the [Jastrow](#) Factor at the walker's position.

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

**3.25.2.11** `double Jastrow::get_variational_derivative ( const Walker * walker, int n )` [protected], [virtual]

Calculates the derivative of the [Jastrow](#) factor with respect to a variational parameter.

## Parameters

<i>n</i>	The index of the variational parameter for which the derivative is to be taken
<i>walker</i>	The walker holds the positions etc. needed to evaluate the derivative

Reimplemented in [No\\_Jastrow](#), and [Pade\\_Jastrow](#).

**3.25.2.12** `virtual void Jastrow::initialize ( )` [pure virtual]

Initializes the non-variational parameters needed by the [Jastrow](#) Factor.

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

3.25.2.13 `virtual void Jastrow::set_parameter ( double param, int n )` `[protected], [pure virtual]`

Sets variational parameters.

#### Parameters

<i>n</i>	The index of the sought variational parameter
<i>param</i>	The new value of parameter [n]

Implemented in [Pade\\_Jastrow](#), and [No\\_Jastrow](#).

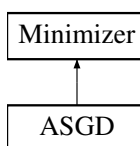
The documentation for this class was generated from the following files:

- `src/Jastrow/Jastrow.h`
- `src/Jastrow/Jastrow.cpp`

## 3.26 Minimizer Class Reference

Class for minimization methods used to obtain optimal variational parameters.

Inheritance diagram for Minimizer:



### Public Member Functions

- [Minimizer](#) (`VMC *vmc`, const [ParParams](#) &, const arma::rowvec &alpha, const arma::rowvec &beta)  
*Constructor.*
- [Orbitals](#) \* `get_orbitals` ()
- [Jastrow](#) \* `get_jastrow` ()
- virtual void `minimize` ()=0  
*Method for executing the minimization main solver.*
- void `output` (std::string message, double number=-1)  
*Method for dumping variational parameter values to screen.*

### Protected Member Functions

- virtual void `update_parameters` ()=0  
*Method for updating the variational parameters based on the previous step.*

### Protected Attributes

- int `n_nodes`
- bool `is_master`
- [VMC](#) \* `vmc`  
*Uses [VMC](#) methods to calculate stochastic variational gradients.*

- [STDOUT](#) \* [std\\_out](#)  
*Output object. Wraps and replaces `std::cout`.*
- `std::stringstream` [s](#)
- `int` [Nspatial\\_params](#)  
*The number of variational parameters in the spatial trial wave function.*
- `int` [Njastrow\\_params](#)  
*The number of variational parameters in the [Jastrow](#) factor.*
- `int` [Nparams](#)  
*The total number of variational parameters.*

### 3.26.1 Detailed Description

Class for minimization methods used to obtain optimal variational parameters.

### 3.26.2 Constructor & Destructor Documentation

3.26.2.1 `Minimizer::Minimizer ( VMC * vmc, const ParParams & pp, const arma::rowvec & alpha, const arma::rowvec & beta )`

Constructor.

Parameters

<i>vmc</i>	The <a href="#">VMC</a> object used for storing variational parameters and calculating stochastic gradients.
<i>alpha</i>	Vector of initial conditions of spatial variational parameters
<i>beta</i>	Vector of initial conditions of <a href="#">Jastrow</a> variational parameters

### 3.26.3 Member Function Documentation

3.26.3.1 `virtual void Minimizer::update_parameters ( )` [`protected`], [`pure virtual`]

Method for updating the variational parameters based on the previous step.

Needs to be implemented by a subclass.

Implemented in [ASGD](#).

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

- `src/Minimizer/Minimizer.h`
- `src/Minimizer/Minimizer.cpp`

## 3.27 MinimizerParams Struct Reference

Struct used to initialize Minimization parameters.

### Public Attributes

- `double` [max\\_step](#)
- `double` [f\\_max](#)
- `double` [f\\_min](#)
- `double` [omega](#)
- `double` [A](#)

- double **a**
- int **SGDsamples**
- int **n\_w**
- int **therm**
- int **n\_c**
- int **n\_c\_SGD**
- arma::rowvec **alpha**  
*Initial condition for the spatial variational parameter(s).*
- arma::rowvec **beta**  
*Initial condition for the *Jastrow* variational parameter(s).*

### 3.27.1 Detailed Description

Struct used to initialize Minimization parameters.

See Also

[ASGD](#)

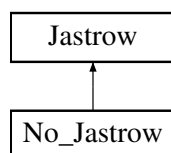
The documentation for this struct was generated from the following file:

- src/structs.h

## 3.28 No\_Jastrow Class Reference

Class loaded when no correlation factor is used.

Inheritance diagram for No\_Jastrow:



### Public Member Functions

- void **get\_grad** ([Walker](#) \*walker) const
- void **get\_grad** (const [Walker](#) \*walker\_pre, [Walker](#) \*walker\_post, int i) const  
*Updates the gradient for a new particle move.*
- void **initialize** ()
- void **get\_dJ\_matrix** ([Walker](#) \*walker, int i) const  
*Updates the summation factors of *Jastrow* factors closed form expressions.*
- double **get\_j\_ratio** (const [Walker](#) \*walker\_post, const [Walker](#) \*walker\_pre, int i) const  
*Calculates the ratio of the *Jastrow* factor needed by metropolis.*
- double **get\_val** (const [Walker](#) \*walker) const
- double **get\_lapl\_sum** ([Walker](#) \*walker) const  
*Method for calculating the Laplacian.*

## Protected Member Functions

- double [get\\_parameter](#) (int n)  
*Returns variational parameters.*
- void [set\\_parameter](#) (double param, int n)  
*Sets variational parameters.*
- double [get\\_variational\\_derivative](#) (const [Walker](#) \*walker, int n)  
*Calculates the derivative of the [Jastrow](#) factor with respect to a variational parameter.*

## Additional Inherited Members

### 3.28.1 Detailed Description

Class loaded when no correlation factor is used.

### 3.28.2 Member Function Documentation

**3.28.2.1** void No\_Jastrow::get\_dj\_matrix ( [Walker](#) \* walker, int i ) const [inline],[virtual]

Updates the summation factors of [Jastrow](#) factors closed form expressions.

Used to optimize the calculations as few of these terms change as we move a particle.

#### Parameters

<i>i</i>	Particle number.
----------	------------------

Implements [Jastrow](#).

**3.28.2.2** void No\_Jastrow::get\_grad ( [Walker](#) \* walker ) const [inline],[virtual]

Calculates the entire Cartesian gradient.

Implements [Jastrow](#).

**3.28.2.3** void No\_Jastrow::get\_grad ( const [Walker](#) \* walker\_pre, [Walker](#) \* walker\_post, int i ) const [inline],[virtual]

Updates the gradient for a new particle move.

#### Parameters

<i>walker_post</i>	<a href="#">Walker</a> at current time step
<i>walker_pre</i>	<a href="#">Walker</a> at previous time step
<i>i</i>	Particle number.

Implements [Jastrow](#).

**3.28.2.4** double No\_Jastrow::get\_j\_ratio ( const [Walker](#) \* walker\_new, const [Walker](#) \* walker\_old, int i ) const [inline],[virtual]

Calculates the ratio of the [Jastrow](#) factor needed by metropolis.

## Parameters

<i>walker_new</i>	<a href="#">Walker</a> at current time step
<i>walker_old</i>	<a href="#">Walker</a> at previous time step
<i>i</i>	The particle number.

Implements [Jastrow](#).

**3.28.2.5** `double No_Jastrow::get_lapl_sum ( Walker * walker ) const` `[inline],[virtual]`

Method for calculating the Laplacian.

Calculates the sum of all particles Laplacians.

Implements [Jastrow](#).

**3.28.2.6** `double No_Jastrow::get_parameter ( int n )` `[inline],[protected],[virtual]`

Returns variational parameters.

## Parameters

<i>n</i>	The index of the sought variational parameter
----------	---

## Returns

Variational parameter with index [n]

Implements [Jastrow](#).

**3.28.2.7** `double No_Jastrow::get_val ( const Walker * walker ) const` `[inline],[virtual]`

Calculates the value of the [Jastrow](#) Factor at the walker's position.

Implements [Jastrow](#).

**3.28.2.8** `double No_Jastrow::get_variational_derivative ( const Walker * walker, int n )` `[inline],[protected],[virtual]`

Calculates the derivative of the [Jastrow](#) factor with respect to a variational parameter.

## Parameters

<i>n</i>	The index of the variational parameter for which the derivative is to be taken
<i>walker</i>	The walker holds the positions etc. needed to evaluate the derivative

Reimplemented from [Jastrow](#).

**3.28.2.9** `void No_Jastrow::initialize ( )` `[inline],[virtual]`

Initializes the non-variational parameters needed by the [Jastrow](#) Factor.

Implements [Jastrow](#).

3.28.2.10 `void No_Jastrow::set_parameter ( double param, int n )` `[inline]`, `[protected]`, `[virtual]`

Sets variational parameters.

#### Parameters

<i>n</i>	The index of the sought variational parameter
<i>param</i>	The new value of parameter [ <i>n</i> ]

Implements [Jastrow](#).

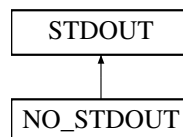
The documentation for this class was generated from the following files:

- `src/Jastrow/No_Jastrow/No_Jastrow.h`
- `src/Jastrow/No_Jastrow/No_Jastrow.cpp`

## 3.29 NO\_STDOUT Class Reference

Class for suppressing standard output. Every node but the master has this. If-tests around `cout` is avoided.

Inheritance diagram for NO\_STDOUT:



### Public Member Functions

- virtual void **cout** (std::stringstream &a)

### 3.29.1 Detailed Description

Class for suppressing standard output. Every node but the master has this. If-tests around `cout` is avoided.

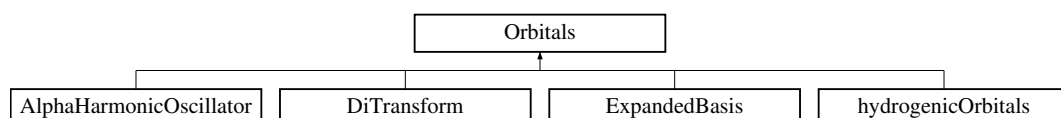
The documentation for this class was generated from the following file:

- `src/structs.h`

## 3.30 Orbitals Class Reference

Superclass for the single particle orbital classes. Handles everything specific regarding choice of single particle basis.

Inheritance diagram for Orbitals:



## Public Member Functions

- **Orbitals** (int n\_p, int dim)
- virtual void **set\_qnum\_indie\_terms** ([Walker](#) \*walker, int i)  
*Calculates single particle wave function terms which are independent of the quantum numbers.*
- virtual double **phi** (const [Walker](#) \*walker, int particle, int q\_num)  
*Calculates the single particle wave function for a given walker's particle.*
- virtual double **del\_phi** (const [Walker](#) \*walker, int particle, int q\_num, int d)  
*Calculates the single particle wave function derivative for a given walker's particle and dimension.*
- virtual double **lapl\_phi** (const [Walker](#) \*walker, int particle, int q\_num)  
*Calculates the single particle wave function for a given walker's particle.*
- void **set\_qmc\_ptr** ([QMC](#) \*qmc)

## Protected Member Functions

- virtual double **get\_parameter** (int n)=0  
*A method for retrieving variational parameters.*
- virtual void **set\_parameter** (double parameter, int n)=0  
*A method for setting variational parameters.*
- double **get\_variational\_derivative** ([Walker](#) \*walker)  
*A method for calculating the variational derivative.*
- virtual double **get\_dell\_alpha\_phi** ([Walker](#) \*walker, int p, int q\_num)
- double **num\_diff** (const [Walker](#) \*walker, int particle, int q\_num, int d)  
*Method for calculating the single particle derivative using a finite difference scheme.*
- double **num\_ddiff** (const [Walker](#) \*walker, int particle, int q\_num)  
*Method for calculating the single particle Laplacian using a finite difference scheme.*
- void **testLaplace** (const [Walker](#) \*walker, int particle, int q\_num)  
*Method for validating closed form expressions for laplacians by comparing them to numerical calculations.*
- void **testDell** (const [Walker](#) \*walker, int particle, int q\_num, int d)  
*Method for validating closed form expressions for derivatives by comparing them to numerical calculations.*
- virtual double **get\_coulomb\_element** (const arma::uvec &qnum\_set)  
*Method for calculating the anti-symmetrized Coulomb matrix elements.*
- virtual double **get\_sp\_energy** (int qnum) const

## Protected Attributes

- int **n\_p**
- int **n2**
- int **dim**
- int **max\_implemented**  
*The maximum number basis size supported for any system ##RYDD OPP.*
- [QMC](#) \* **qmc**  
*A pointer to the [QMC](#) solver object. Needed for numerical variational derivatives.*
- double **h**  
*The step length for finite difference schemes.*
- double **h2**
- double **two\_h**
- arma::imat **qnums**  
*Quantum number matrix needed by Hartree-Fock and the variational derivatives.*
- [BasisFunctions](#) \*\* **basis\_functions**  
*A vector mapping a quantum number index to a single particle wave function.*



- [BasisFunctions](#) \*\*\* [dell\\_basis\\_functions](#)  
A maxtrix maping a quantum number- and dimension index to a single particle wave function derivative.
- [BasisFunctions](#) \*\* [lapl\\_basis\\_functions](#)  
A vector maping a quantum number index to a single particle wave function Laplacian.

## Friends

- class **HartreeFock**
- class **Minimizer**
- class **ASGD**
- class **stdoutASGD**
- class **DiTransform**

### 3.30.1 Detailed Description

Superclass for the single particle orbital classes. Handles everything specific regarding choice of single particle basis.

### 3.30.2 Member Function Documentation

3.30.2.1 `double Orbitals::del_phi ( const Walker * walker, int particle, int q_num, int d )` [virtual]

Calculates the single particle wave function derivative for a given walker's particle and dimension.

#### Parameters

<i>q_num</i>	The quantum number index.
<i>d</i>	The dimension for which the derivative should be calculated (x,y,z).

Reimplemented in [DiTransform](#), and [ExpandedBasis](#).

3.30.2.2 `double Orbitals::get_coulomb_element ( const arma::uvec & qnum_set )` [protected], [virtual]

Method for calculating the anti-symmetrized Coulumb matrix elements.

Used by Hartree Fock

Reimplemented in [AlphaHarmonicOscillator](#), and [hydrogenicOrbitals](#).

3.30.2.3 `virtual double Orbitals::get_parameter ( int n )` [protected], [pure virtual]

A method for retrieving variational parameters.

#### Parameters

<i>n</i>	Index of the sought variational parameter.
----------	--

Implemented in [AlphaHarmonicOscillator](#), [hydrogenicOrbitals](#), and [DiTransform](#).

3.30.2.4 `double Orbitals::get_variational_derivative ( Walker * walker )` [protected]

A method for calculating the variational derivative.

By default uses a finite difference scheme. Can be overridden to evaluate a closed form expression.

## Parameters

<i>n</i>	Index of the sought variational parameter.
----------	--

3.30.2.5 `double Orbitals::lapl_phi ( const Walker * walker, int particle, int q_num )` [virtual]

Calculates the single particle wave function for a given walker's particle.

## Parameters

<i>q_num</i>	The quantum number index.
--------------	---------------------------

Reimplemented in [DiTransform](#), and [ExpandedBasis](#).

3.30.2.6 `double Orbitals::num_ddiff ( const Walker * walker, int particle, int q_num )` [protected]

Method for calculating the single particle Laplacian using a finite difference scheme.

Method [lapl\\_phi\(\)](#) can be overridden to use this method in case no closed form expressions are implemented.

## Parameters

<i>q_num</i>	The quantum number index.
--------------	---------------------------

3.30.2.7 `double Orbitals::num_diff ( const Walker * walker, int particle, int q_num, int d )` [protected]

Method for calculating the single particle derivative using a finite difference scheme.

Method [del\\_phi\(\)](#) can be overridden to use this method in case no closed form expressions are implemented.

## Parameters

<i>d</i>	The dimension for which the derivative should be calculated (x,y,z).
----------	--

3.30.2.8 `double Orbitals::phi ( const Walker * walker, int particle, int q_num )` [virtual]

Calculates the single particle wave function for a given walker's particle.

## Parameters

<i>q_num</i>	The quantum number index.
--------------	---------------------------

Reimplemented in [DiTransform](#), and [ExpandedBasis](#).

3.30.2.9 `virtual void Orbitals::set_parameter ( double parameter, int n )` [protected], [pure virtual]

A method for setting variational parameters.

## Parameters

<i>n</i>	Index of the sought variational parameter.
<i>parameter</i>	The new value of the variational parameter.

Implemented in [AlphaHarmonicOscillator](#), [hydrogenicOrbitals](#), and [DiTransform](#).

3.30.2.10 `virtual void Orbitals::set_qnum_indie_terms ( Walker * walker, int i ) [inline], [virtual]`

Calculates single particle wave function terms which are independent of the quantum numbers.

If a term in the single particle functions are independent of the quantum number, this function can be overridden to calculate them beforehand (for each particle), and rather extract the value instead of recalculating.

#### Parameters

<code>i</code>	Particle number.
----------------	------------------

Reimplemented in [AlphaHarmonicOscillator](#), [hydrogenicOrbitals](#), [DiTransform](#), and [ExpandedBasis](#).

3.30.2.11 `void Orbitals::testDell ( const Walker * walker, int particle, int q_num, int d ) [protected]`

Method for validating closed form expressions for derivatives by comparing them to numerical calculations.

#### Parameters

<code>q_num</code>	The quantum number index.
<code>d</code>	The dimension for which the derivative should be calculated (x,y,z).

3.30.2.12 `void Orbitals::testLaplace ( const Walker * walker, int particle, int q_num ) [protected]`

Method for validating closed form expressions for laplacians by comparing them to numerical calculations.

#### Parameters

<code>q_num</code>	The quantum number index.
--------------------	---------------------------

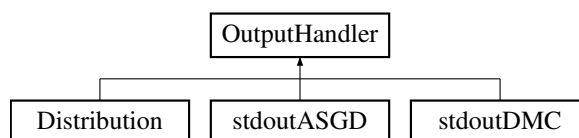
The documentation for this class was generated from the following files:

- `src/Orbitals/Orbitals.h`
- `src/Orbitals/Orbitals.cpp`

## 3.31 OutputHandler Class Reference

Class for handling output-methods. Designed to avoid rewriting code, as well as avoid if-tests if output is not desired.

Inheritance diagram for OutputHandler:



### Public Member Functions

- [OutputHandler](#) (std::string filename, std::string path, bool parallel, int node, int n\_nodes)  
*Constructor.*
- virtual void [dump](#) ()=0  
*Methods for updating the output.*

- virtual void `finalize` ()  
*Finalizes the output.*

### Protected Member Functions

- void `init_file` ()

### Protected Attributes

- bool `parallel`
- int `node`
- int `n_nodes`
- bool `use_file`  
*If `init_file()` is called, this flag is set true. Assures correct finalization.*
- std::stringstream `s`
- std::string `filename`
- std::string `path`
- std::ofstream `file`

#### 3.31.1 Detailed Description

Class for handling output-methods. Designed to avoid rewriting code, as well as avoid if-tests if output is not desired.

See Also

QMC::output\_handler, Minimizer::output\_handler

#### 3.31.2 Constructor & Destructor Documentation

3.31.2.1 `OutputHandler::OutputHandler ( std::string filename, std::string path, bool parallel, int node, int n_nodes )`

Constructor.

Parameters

<i>filename</i>	The name of the output file.
<i>path</i>	The path of the output.

#### 3.31.3 Member Function Documentation

3.31.3.1 `virtual void OutputHandler::dump ( ) [pure virtual]`

Methods for updating the output.

Typically retrieves information through the solver pointers (given correct accessibility levels/friend)

Implemented in [Distribution](#), [stdoutDMC](#), and [stdoutASGD](#).

3.31.3.2 `void OutputHandler::finalize ( ) [virtual]`

Finalizes the output.

Closes file if `use_file` flag is true. Can be overridden if more complex tasks needs to be done, such as calculating histograms etc.

See Also

[Distribution::finalize\(\)](#)

Reimplemented in [Distribution](#).

### 3.31.3.3 void OutputHandler::init\_file ( ) [protected]

Opens a file with filename at path supplied in constructor. Subclass implementations can call this function. Super-class does not.

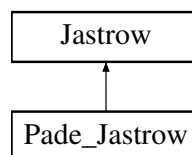
The documentation for this class was generated from the following files:

- src/OutputHandler/OutputHandler.h
- src/OutputHandler/OutputHandler.cpp

## 3.32 Pade\_Jastrow Class Reference

The Pade [Jastrow](#) factor with a single variational parameter.

Inheritance diagram for Pade\_Jastrow:



### Public Member Functions

- **Pade\_Jastrow** ([GeneralParams](#) &, [VariationalParams](#) &)
- void [initialize](#) ()
- void [get\\_grad](#) ([Walker](#) \*walker) const
- void [get\\_grad](#) (const [Walker](#) \*walker\_pre, [Walker](#) \*walker\_post, int i) const  
*Updates the gradient for a new particle move.*
- void [get\\_dJ\\_matrix](#) ([Walker](#) \*walker, int i) const  
*Updates the summation factors of [Jastrow](#) factors closed form expressions.*
- double [get\\_j\\_ratio](#) (const [Walker](#) \*walker\_new, const [Walker](#) \*walker\_old, int i) const  
*Calculates the ratio of the [Jastrow](#) factor needed by metropolis.*
- double [get\\_val](#) (const [Walker](#) \*walker) const
- double [get\\_lapl\\_sum](#) ([Walker](#) \*walker) const  
*Method for calculating the Laplacian.*

### Protected Member Functions

- double [get\\_variational\\_derivative](#) (const [Walker](#) \*walker, int n)  
*Calculates the derivative of the [Jastrow](#) factor with respect to a variational parameter.*
- void [set\\_parameter](#) (double param, int n)  
*Sets variational parameters.*
- double [get\\_parameter](#) (int n)  
*Returns variational parameters.*

## Protected Attributes

- double [beta](#)  
*The variational parameter.*
- arma::mat [a](#)  
*The spin-dependent variables taking care of the cusp condition.*

### 3.32.1 Detailed Description

The Pade [Jastrow](#) factor with a single variational parameter.

### 3.32.2 Member Function Documentation

**3.32.2.1** void Pade\_Jastrow::get\_dJ\_matrix ( Walker \* *walker*, int *i* ) const [virtual]

Updates the summation factors of [Jastrow](#) factors closed form expressions.  
Used to optimize the calculations as few of these terms change as we move a particle.

#### Parameters

<i>i</i>	Particle number.
----------	------------------

Implements [Jastrow](#).

**3.32.2.2** void Pade\_Jastrow::get\_grad ( Walker \* *walker* ) const [virtual]

Calculates the entire Cartesian gradient.  
Implements [Jastrow](#).

**3.32.2.3** void Pade\_Jastrow::get\_grad ( const Walker \* *walker\_pre*, Walker \* *walker\_post*, int *i* ) const [virtual]

Updates the gradient for a new particle move.

#### Parameters

<i>walker_post</i>	<a href="#">Walker</a> at current time step
<i>walker_pre</i>	<a href="#">Walker</a> at previous time step
<i>i</i>	Particle number.

Implements [Jastrow](#).

**3.32.2.4** double Pade\_Jastrow::get\_j\_ratio ( const Walker \* *walker\_new*, const Walker \* *walker\_old*, int *i* ) const [virtual]

Calculates the ratio of the [Jastrow](#) factor needed by metropolis.

#### Parameters

<i>walker_new</i>	<a href="#">Walker</a> at current time step
<i>walker_old</i>	<a href="#">Walker</a> at previous time step
<i>i</i>	The particle number.

Implements [Jastrow](#).

3.32.2.5 `double Pade_Jastrow::get_lapl_sum ( Walker * walker ) const` [virtual]

Method for calculating the Laplacian.

Calculates the sum of all particles Laplacians.

Implements [Jastrow](#).

3.32.2.6 `double Pade_Jastrow::get_parameter ( int n )` [inline],[protected],[virtual]

Returns variational parameters.

#### Parameters

<i>n</i>	The index of the sought variational parameter
----------	---

#### Returns

Variational parameter with index [n]

Implements [Jastrow](#).

3.32.2.7 `double Pade_Jastrow::get_val ( const Walker * walker ) const` [virtual]

Calculates the value of the [Jastrow](#) Factor at the walker's position.

Implements [Jastrow](#).

3.32.2.8 `double Pade_Jastrow::get_variational_derivative ( const Walker * walker, int n )` [protected],[virtual]

Calculates the derivative of the [Jastrow](#) factor with respect to a variational parameter.

#### Parameters

<i>n</i>	The index of the variational parameter for which the derivative is to be taken
<i>walker</i>	The walker holds the positions etc. needed to evaluate the derivative

Reimplemented from [Jastrow](#).

3.32.2.9 `void Pade_Jastrow::initialize ( )` [virtual]

In case of Pade [Jastrow](#), initializing means setting up the a matrix.

Implements [Jastrow](#).

3.32.2.10 `void Pade_Jastrow::set_parameter ( double param, int n )` [inline],[protected],[virtual]

Sets variational parameters.

#### Parameters

<i>n</i>	The index of the sought variational parameter
<i>param</i>	The new value of parameter [n]

Implements [Jastrow](#).

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

- `src/Jastrow/Pade_Jastrow/Pade_Jastrow.h`
- `src/Jastrow/Pade_Jastrow/Pade_Jastrow.cpp`

### 3.33 ParParams Struct Reference

Struct used to initialize parallelization parameters.

#### Public Attributes

- bool `is_master`  
*True for the master node.*
- bool `parallel`  
*True if `n_nodes > 1`.*
- int `node`  
*The process' rank.*
- int `n_nodes`  
*The total number of processes.*

#### 3.33.1 Detailed Description

Struct used to initialize parallelization parameters.

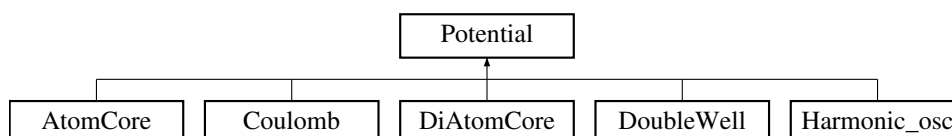
The documentation for this struct was generated from the following file:

- `src/structs.h`

### 3.34 Potential Class Reference

Superclass for potentials. Potentials are stores in a vector in the system object.

Inheritance diagram for Potential:



#### Public Member Functions

- **Potential** (int n\_p, int dim)
- virtual double `get_pot_E` (const `Walker` \*walker) const =0  
*Method for calculating a walker's potential energy.*
- std::string `get_name` ()

#### Public Attributes

- `Sampler` `pot_sampler`



## Protected Attributes

- int **n\_p**
- int **dim**
- std::string **name**

### 3.34.1 Detailed Description

Superclass for potentials. Potentials are stores in a vector in the system object.

See Also

[System::potentials](#), [System::get\\_potential\\_energy\(\)](#)

### 3.34.2 Member Function Documentation

3.34.2.1 `virtual double Potential::get_pot_E ( const Walker * walker ) const` [pure virtual]

Method for calculating a walker's potential energy.

Method overridden by subclasses.

Parameters

<i>walker</i>	The walker for which the potential energy should be calculated.
---------------	---

Returns

The potential energy.

Implemented in [Harmonic\\_osc](#), [AtomCore](#), [Coulomb](#), [DiAtomCore](#), and [DoubleWell](#).

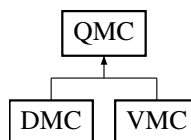
The documentation for this class was generated from the following files:

- src/Potential/Potential.h
- src/Potential/Potential.cpp

## 3.35 QMC Class Reference

The **QMC** superclass. Holds implementations of general functions for both **VMC** and **DMC** in order to avoid rewriting code and emphasize the similarities.

Inheritance diagram for QMC:



## Public Member Functions

- **QMC** ([GeneralParams](#) &, int **n\_c**, [SystemObjects](#) &, [ParParams](#) &, int **n\_w**, int K=1)  
*Constructor.*
- virtual void **run\_method** ()=0

- Method used for executing the solver.*
- void `get_QF` (`Walker *walker`) const
- Method for calculating the Quantum Force.*
- void `get_gradients` (const `Walker *walker_pre`, `Walker *walker_post`, int particle) const
- Method for calculating the gradients after moving a particle.*
- void `get_gradients` (`Walker *walker`) const
- Method for calculating the full gradients.*
- void `get_laplsum` (`Walker *walker`) const
- Method for calculating the Laplacian of all walkers.*
- double `get_wf_value` (const `Walker *walker`) const
- Method for calculating the wave functions value at a given walker's position.*
- double `calculate_local_energy` (const `Walker *walker`)
- Method for calculating the local energy.*
- void `get_accepted_ratio` ()
- Method for calculating the acceptance ratio.*
- void `clean` ()
- Cleans up initializations which distorts successive use of objects.*
- void `set_error_estimator` (`ErrorEstimator *error_estimator`)
- Method for setting the error estimator.*
- virtual void `output` ()=0
- Method for standard output.*
- `System * get_system_ptr` () const
- `Sampling * get_sampling_ptr` () const
- `Jastrow * get_jastrow_ptr` () const
- `Orbitals * get_orbitals_ptr` () const

## Protected Member Functions

- virtual void `set_trial_positions` ()=0
- Method for setting the trial position of the QMC method's walkers.*
- void `diffuse_walker` (`Walker *original`, `Walker *trial`)
- Method for diffusing a walker one time step.*
- double `get_acceptance_ratio` (const `Walker *walker_pre`, const `Walker *walker_post`, int particle) const
- Method for calculating the acceptance ratio used in the Metropolis test.*
- virtual bool `move_authorized` (double A)=0
- Method for deciding whether or not to accept a move.*
- bool `metropolis_test` (double A)
- Method for performing the metropolis test after when diffusing a walker.*
- void `update_walker` (`Walker *walker_pre`, const `Walker *walker_post`, int particle) const
- Method for updating the walker after an accepted step.*
- void `reset_walker` (const `Walker *walker_pre`, `Walker *walker_post`, int particle) const
- Method for resetting the walker after a rejected step.*
- void `copy_walker` (const `Walker *parent`, `Walker *child`) const
- Method for (hard) copying a walker object.*
- void `calculate_energy_necessities` (`Walker *walker`) const
- Method for calculating the necessary quantities needed in order to calculate the local energy.*
- double `get_KE` (const `Walker *walker`)
- Method for calculating the kinetic energy of a walker.*
- void `update_subsamples` (double weight=1.0)
- void `push_subsamples` ()
- void `dump_subsamples` (bool mean\_of\_means=false)

- virtual void [save\\_distribution](#) ()=0  
*Method for storing positional data.*
- virtual void [node\\_comm](#) ()=0  
*Method for performing node communication.*
- void [finalize\\_distribution](#) ()  
*Method for calculating the distribution.*
- void [estimate\\_error](#) () const
- void [set\\_spin\\_state](#) (int particle) const
- void [test\\_ratios](#) (const [Walker](#) \*walker\_pre, const [Walker](#) \*walker\_post, int particle, double R\_qmc) const  
*Method used for testing the optimized ratio calculation.*
- void [test\\_gradients](#) ([Walker](#) \*walker)  
*Method for testing the optimized gradients calculation.*

### Protected Attributes

- [STDOUT](#) \* [std\\_out](#)  
*Output object. Wraps and replaces std::cout.*
- std::stringstream [s](#)
- int [output\\_tresh](#)
- int [n\\_w\\_size](#)  
*The total number of allocated walkers.*
- std::string [runpath](#)  
*The directory which the simulation is set to run.*
- std::string [dist\\_path](#)  
*The path where the distribution are saved.*
- arma::mat [dist](#)  
*Matrix holding positional data for the distribution.*
- int [last\\_inserted](#)  
*Index of last inserted positional data.*
- int [dist\\_tresh](#)  
*Amount of cycles to skip in between storing position data.*
- bool [is\\_master](#)
- bool [parallel](#)
- int [node](#)
- int [n\\_nodes](#)
- int [p\\_start](#)
- int [n\\_c](#)  
*The number of Monte-Carlo cycles.*
- int [thermalization](#)  
*The number of thermalization steps.*
- int [cycle](#)
- int [n\\_w](#)  
*The number of walkers.*
- int [n\\_p](#)
- int [n2](#)
- int [dim](#)
- unsigned long int [accepted](#)  
*Number of accepted moves.*
- unsigned long int [total\\_samples](#)  
*Total number of moves.*
- double [local\\_E](#)

- The last calculated local energy.*
- Walker \* [trial\\_walker](#)  
*The trial walker used to test a move.*
- Walker \*\* [original\\_walkers](#)  
*A list of  $n_w$  walkers used in DMC.*
- Jastrow \* [jastrow](#)  
*The Jastrow object.*
- Sampling \* [sampling](#)  
*The Sampling object.*
- System \* [system](#)  
*The system object.*
- ErrorEstimator \* [error\\_estimator](#)  
*The error estimator.*
- Sampler [kinetic\\_sampler](#)
- Distribution \* [distribution](#)

### 3.35.1 Detailed Description

The [QMC](#) superclass. Holds implementations of general functions for both [VMC](#) and [DMC](#) in order to avoid rewriting code and emphasize the similarities.

### 3.35.2 Constructor & Destructor Documentation

3.35.2.1 `QMC::QMC ( GeneralParams & gP, int n_c, SystemObjects & sO, ParParams & pp, int n_w, int K = 1 )`

Constructor.

K K times  $n_w$  walkers are initialized.  $K \neq 0$  only sensible in [DMC](#).

### 3.35.3 Member Function Documentation

3.35.3.1 `void QMC::calculate_energy_necessities ( Walker * walker ) const` [protected]

Method for calculating the necessary quantities needed in order to calculate the local energy.

See Also

[Sampling::calculate\\_energy\\_necessities\(\)](#)

3.35.3.2 `double QMC::calculate_local_energy ( const Walker * walker )`

Method for calculating the local energy.

See Also

[get\\_KE\(\)](#), [System::get\\_potential\\_energy\(\)](#)

3.35.3.3 `void QMC::copy_walker ( const Walker * parent, Walker * child ) const` [protected]

Method for (hard) copying a walker object.

Parameters

<i>parent, child</i>	The parent is copied to the child.
----------------------	------------------------------------

**3.35.3.4** void QMC::diffuse\_walker ( Walker \* *original*, Walker \* *trial* ) [protected]

Method for diffusing a walker one time step.

The trial walker must equal the original walker in input. The original walker is updated on output.

**3.35.3.5** void QMC::estimate\_error ( ) const [protected]

Estimates and finalizes the [ErrorEstimator](#) object initialized in the error\_estimator vector.

**3.35.3.6** void QMC::get\_gradients ( const Walker \* *walker\_pre*, Walker \* *walker\_post*, int *particle* ) const

Method for calculating the gradients after moving a particle.

See Also

[Jastrow::get\\_grad\(\)](#), [System::get\\_spatial\\_grad\(\)](#)

**3.35.3.7** void QMC::get\_gradients ( Walker \* *walker* ) const

Method for calculating the full gradients.

See Also

[Jastrow::get\\_grad\(\)](#), [System::get\\_spatial\\_grad\(\)](#)

**3.35.3.8** void QMC::get\_lapsum ( Walker \* *walker* ) const

Method for calculating the Laplacian of all walkers.

See Also

[System::get\\_spatial\\_lapl\\_sum\(\)](#), [Jastrow::get\\_lapl\\_sum\(\)](#)

**3.35.3.9** double QMC::get\_wf\_value ( const Walker \* *walker* ) const

Method for calculating the wave functions value at a given walker's position.

See Also

[System::get\\_spatial\\_wf\(\)](#), [Jastrow::get\\_val\(\)](#)

**3.35.3.10** bool QMC::metropolis\_test ( double *A* ) [protected]

Method for performing the metropolis test after when diffusing a walker.

Parameters

<i>A</i>	The acceptance ratio calculated by <a href="#">get_acceptance_ratio()</a> .
----------	---

**3.35.3.11** `virtual bool QMC::move_authorized ( double A ) [protected],[pure virtual]`

Method for deciding whether or not to accept a move.

Wraps the metropolis sampling with possibilities of overriding.

See Also

[System::allow\\_transition\(\)](#)

Implemented in [DMC](#), and [VMC](#).

**3.35.3.12** `void QMC::reset_walker ( const Walker * walker_pre, Walker * walker_post, int particle ) const [protected]`

Method for resetting the walker after a rejected step.

Given a particle number, the method only resets the changed values.

Parameters

<i>walker_post</i>	<a href="#">Walker</a> at current time step
<i>walker_pre</i>	<a href="#">Walker</a> at previous time step

**3.35.3.13** `virtual void QMC::save_distribution ( ) [protected],[pure virtual]`

Method for storing positional data.

Stored in the dist matrix. Used by `OutputHandler::Distribution`.

See Also

[Distribution::dump\(\)](#), [VMC::save\\_distribution\(\)](#), [DMC::save\\_distribution\(\)](#)

Implemented in [DMC](#), and [VMC](#).

**3.35.3.14** `void QMC::set_spin_state ( int particle ) const [protected]`

Since the spatial wave function is split, certain values are unchanged if the moved particle has opposite spin. Assuming a two-level system, the first half of the particles are assumed to have one spin value, and the second half the other.

This method sets the start and end position of the block that needs to be changed.

See Also

[System::start](#), [System::end](#)

**3.35.3.15** `virtual void QMC::set_trial_positions ( ) [protected],[pure virtual]`

Method for setting the trial position of the [QMC](#) method's walkers.

See Also

[Sampling::set\\_trial\\_pos\(\)](#)

Implemented in [DMC](#), and [VMC](#).

**3.35.3.16** `void QMC::test_gradients ( Walker * walker )` `[protected]`

Method for testing the optimized gradients calculation.

Compares with finite difference calculation.

**3.35.3.17** `void QMC::test_ratios ( const Walker * walker_pre, const Walker * walker_post, int particle, double R_qmc )`  
`const` `[protected]`

Method used for testing the optimized ratio calculation.

Compares to brute force computation of the wave function values. `R_qmc` The optimized trial wave function ratio (spatial and [Jastrow](#)).

**3.35.3.18** `void QMC::update_walker ( Walker * walker_pre, const Walker * walker_post, int particle )` `const`  
`[protected]`

Method for updating the walker after an accepted step.

Given a particle number, the method only updates the changed values.

#### Parameters

<code>walker_post</code>	<a href="#">Walker</a> at current time step
<code>walker_pre</code>	<a href="#">Walker</a> at previous time step

### 3.35.4 Member Data Documentation

**3.35.4.1** `int QMC::cycle` `[protected]`

The current Monte-Carlo cycle.

**3.35.4.2** `int QMC::n_w` `[protected]`

The number of walkers.

[VMC](#) stores this many cycles in case of [DMC](#)

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

- `src/QMC/QMC.h`
- `src/QMC/QMC.cpp`

## 3.36 Sampler Class Reference

### Public Member Functions

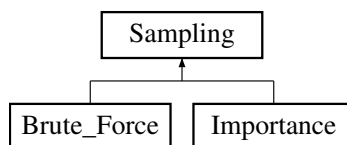
- `void queue_value` (const double &value)
- `void update_mean` (const double &weight)
- `void push_mean` ()
- `const double extract_mean` ()
- `const double extract_mean_of_means` ()

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

- `src/Sampler/Sampler.h`

### 3.37 Sampling Class Reference

Inheritance diagram for Sampling:



#### Public Member Functions

- **Sampling** (int n\_p, int dim)
- void **update\_pos** (const Walker \*walker\_pre, Walker \*walker\_post, int particle) const  
*Method for updating the position of a walker's particle.*
- virtual void **update\_necessities** (const Walker \*walker\_pre, Walker \*walker\_post, int particle) const =0  
*Method for updating the sampling specific necessary values.*
- virtual void **update\_walker** (Walker \*walker\_pre, const Walker \*walker\_post, int particle) const =0
- virtual void **reset\_walker** (const Walker \*walker\_pre, Walker \*walker\_post, int particle) const =0
- void **set\_trial\_pos** (Walker \*walker)  
*Method for setting the trial position for a given walker.*
- void **set\_trial\_states** (Walker \*walker)  
*Method for setting up the single particle orbitals and it's derivatives for a given walker.*
- virtual void **get\_necessities** (Walker \*walker)=0  
*Method for calculating the sampling specific necessary values.*
- virtual void **calculate\_energy\_necessities** (Walker \*walker) const =0  
*Method for calculating the sampling specific necessary values in order to compute the local energy.*
- virtual void **copy\_walker** (const Walker \*parent, Walker \*child) const =0  
*Method for copying the sampling specific parts of a walker.*
- virtual double **get\_g\_ratio** (const Walker \*walker\_post, const Walker \*walker\_pre) const  
*Method for calculating the diffusion Green's function ratios.*
- double **get\_branching\_Gfunc** (double E\_x, double E\_y, double E\_T) const  
*Calculates the Branching Green's function ratio needed by DMC.*
- void **set\_qmc\_ptr** (QMC \*qmc)
- void **set\_dt** (double dt)
- double **get\_dt** () const
- double **get\_std** () const
- double **call\_RNG** ()  
*Calls a uniform random number generator.*
- void **set\_spin\_state** (int start, int end)

#### Protected Member Functions

- void **set\_deadlock** (const double deadlock\_x)  
*Position of the frozen particle. y=z=0.*
- void **clear\_deadlock** ()
- friend **QMC::QMC** (GeneralParams &, int, SystemObjects &, ParParams &, int, int)



## Protected Attributes

- int **n\_p**
- int **n2**
- int **dim**
- int [start](#)
- int [end](#)
- bool **deadlock**
- double [deadlock\\_x](#)  
*If true, a particle is frozen in the simulation.*
- [Diffusion](#) \* [diffusion](#)  
*The [Diffusion](#) object.*
- [QMC](#) \* [qmc](#)  
*The [QMC](#) main solver object. Needed to access e.g. the system object.*

## Friends

- void **QMC::clean** ()

### 3.37.1 Member Function Documentation

#### 3.37.1.1 double Sampling::call\_RNG ( ) [inline]

Calls a uniform random number generator.

Returns a random uniform number on [0,1).

#### 3.37.1.2 virtual void Sampling::copy\_walker ( const Walker \* *parent*, Walker \* *child* ) const [pure virtual]

Method for copying the sampling specific parts of a walker.

See Also

[QMC::copy\\_walker\(\)](#)

Implemented in [Importance](#), and [Brute\\_Force](#).

#### 3.37.1.3 double Sampling::get\_branching\_Gfunc ( double *E\_x*, double *E\_y*, double *E\_T* ) const [inline]

Calculates the Branching Green's function ratio needed by [DMC](#).

Parameters

<i>E_x</i>	Energy at current time step
<i>E_y</i>	Energy at previous time step

Returns

The Branching Green's function ratio

#### 3.37.1.4 virtual double Sampling::get\_g\_ratio ( const Walker \* *walker\_post*, const Walker \* *walker\_pre* ) const [inline],[virtual]

Method for calculating the diffusion Green's function ratios.

See the [Diffusion](#) class for documentation.

**3.37.1.5** `virtual void Sampling::get_necessities ( Walker * walker ) [pure virtual]`

Method for calculating the sampling specific necessary values.

Called after a trial position is set.

Implemented in [Importance](#), and [Brute\\_Force](#).

**3.37.1.6** `virtual void Sampling::reset_walker ( const Walker * walker_pre, Walker * walker_post, int particle ) const [pure virtual]`

See Also

[QMC::reset\\_walker\(\)](#)

Implemented in [Brute\\_Force](#), and [Importance](#).

**3.37.1.7** `void Sampling::set_spin_state ( int start, int end ) [inline]`

See Also

[QMC::set\\_spin\\_state\(\)](#)

**3.37.1.8** `void Sampling::update_pos ( const Walker * walker_pre, Walker * walker_post, int particle ) const`

Method for updating the position of a walker's particle.

Sets a new position according to the diffusion rules, and calls all the functions necessary to get all the values updates, e.g. `System::calc_for_new_pos()`

**3.37.1.9** `virtual void Sampling::update_walker ( Walker * walker_pre, const Walker * walker_post, int particle ) const [pure virtual]`

See Also

[QMC::update\\_walker\(\)](#)

Implemented in [Importance](#), and [Brute\\_Force](#).

## 3.37.2 Member Data Documentation

**3.37.2.1** `Diffusion* Sampling::diffusion [protected]`

The [Diffusion](#) object.

See Also

[Diffusion](#)

**3.37.2.2** `int Sampling::end [protected]`

See Also

[System::end](#)

3.37.2.3 `int Sampling::start` `[protected]`

## See Also

[System::start](#)

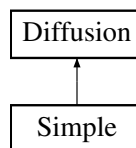
The documentation for this class was generated from the following files:

- `src/Sampling/Sampling.h`
- `src/Sampling/Sampling.cpp`

## 3.38 Simple Class Reference

[Simple](#) Isotropic diffusion model.

Inheritance diagram for Simple:



### Public Member Functions

- **Simple** (int `n_p`, int `dim`, double `timestep`, seed\_type `random_seed`, double `D=0.5`)
- double `get_new_pos` (const [Walker](#) \*`walker`, int `i`, int `j`)  
*Virtual function returning the new position.*
- double `get_g_ratio` (const [Walker](#) \*`walker_post`, const [Walker](#) \*`walker_pre`) const  
*Calculates the [Diffusion](#) Green's function ratio needed by metropolis.*

### Additional Inherited Members

#### 3.38.1 Detailed Description

[Simple](#) Isotropic diffusion model.

#### 3.38.2 Member Function Documentation

3.38.2.1 `double Simple::get_g_ratio ( const Walker * walker_post, const Walker * walker_pre ) const` `[inline]`,  
`[virtual]`

Calculates the [Diffusion](#) Green's function ratio needed by metropolis.

##### Parameters

<code>walker_post</code>	<a href="#">Walker</a> at current time step.
<code>walker_pre</code>	<a href="#">Walker</a> at previous time step.

##### Returns

The [Diffusion](#) Green's function ratio.

Implements [Diffusion](#).

### 3.38.2.2 double Simple::get\_new\_pos ( const Walker \* walker, int i, int j ) [inline],[virtual]

Virtual function returning the new position.

Returns the simple diffusion step if not overridden.

#### Parameters

<i>i</i>	Particle number.
<i>j</i>	dimension (x,y,z).

#### Returns

The new position (relative to the old).

Reimplemented from [Diffusion](#).

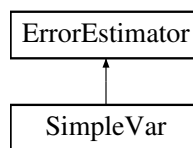
The documentation for this class was generated from the following files:

- src/Diffusion/Simple/Simple.h
- src/Diffusion/Simple/Simple.cpp

## 3.39 SimpleVar Class Reference

Calculates the simple variance of the sampled values.

Inheritance diagram for SimpleVar:



### Public Member Functions

- **SimpleVar** ([ParParams](#) &)
- double [estimate\\_error](#) ()  
*Estimates the error based on the subclass implementation.*
- void [update\\_data](#) (double val)
- void **normalize** ()

### Protected Attributes

- double [f](#)  
*sum variable used to calculate the mean*
- double [f2](#)  
*sum variable used to calculate the mean of squares.*

### Additional Inherited Members

#### 3.39.1 Detailed Description

Calculates the simple variance of the sampled values.

### 3.39.2 Member Function Documentation

#### 3.39.2.1 void SimpleVar::update\_data ( double val ) [virtual]

Overrides the default described in the superclass. Does not store values in memory, but rather use sum variables.

Reimplemented from [ErrorEstimator](#).

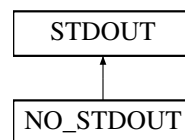
The documentation for this class was generated from the following files:

- src/ErrorEstimator/SimpleVar/SimpleVar.h
- src/ErrorEstimator/SimpleVar/SimpleVar.cpp

## 3.40 STDOUT Class Reference

Class for handling standard output. Only the master node has this object.

Inheritance diagram for STDOUT:



### Public Member Functions

- virtual void **cout** (std::stringstream &a)

#### 3.40.1 Detailed Description

Class for handling standard output. Only the master node has this object.

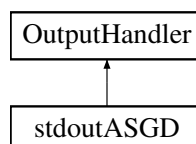
The documentation for this class was generated from the following file:

- src/structs.h

## 3.41 stdoutASGD Class Reference

Class for handling the output of [ASGD](#). Outputs values such as the variational gradients, step length, variational parameters, etc.

Inheritance diagram for stdoutASGD:



### Public Member Functions

- **stdoutASGD** ([ASGD](#) \*asgd, std::string path)

- void `dump` ()

*Methods for updating the output.*

## Additional Inherited Members

### 3.41.1 Detailed Description

Class for handling the output of `ASGD`. Outputs values such as the variational gradients, step length, variational parameters, etc.

### 3.41.2 Member Function Documentation

#### 3.41.2.1 void `stdoutASGD::dump` ( ) [virtual]

Methods for updating the output.

Typically retrieves information through the solver pointers (given correct accessibility levels/friend)

Implements `OutputHandler`.

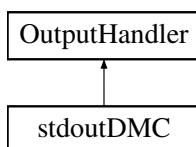
The documentation for this class was generated from the following files:

- `src/OutputHandler/stdoutASGD/stdoutASGD.h`
- `src/OutputHandler/stdoutASGD/stdoutASGD.cpp`

## 3.42 stdoutDMC Class Reference

Class for handling the output of `DMC`. Outputs values such as the trial energy, dmc energy, number of walkers, etc.

Inheritance diagram for `stdoutDMC`:



### Public Member Functions

- **stdoutDMC** (`DMC *dmc`, `std::string path`)
- void `dump` ()

*Methods for updating the output.*

### Protected Attributes

- int `n`  
*Number of times the `dump()` method has been called.*
- double `sumE`  
*Sum of the `DMC` energy used to calculate the trailing average.*
- double `sumN`  
*Sum of the number of walkers used to calculate the trailing average.*
- `DMC * dmc`

## Additional Inherited Members

### 3.42.1 Detailed Description

Class for handling the output of [DMC](#). Outputs values such as the trial energy, dmc energy, number of walkers, etc.

### 3.42.2 Member Function Documentation

#### 3.42.2.1 void stdoutDMC::dump ( ) [virtual]

Methods for updating the output.

Typically retrieves information through the solver pointers (given correct accessibility levels/friend)

Implements [OutputHandler](#).

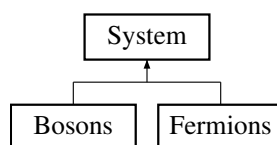
The documentation for this class was generated from the following files:

- src/OutputHandler/stdoutDMC/stdoutDMC.h
- src/OutputHandler/stdoutDMC/stdoutDMC.cpp

## 3.43 System Class Reference

The system class separating [Fermions](#) and [Bosons](#). Designed to generalize the solver in terms of particle species.

Inheritance diagram for System:



## Public Member Functions

- **System** (int n\_p, int dim, [Orbitals](#) \*orbital)
- virtual void [initialize](#) ([Walker](#) \*walker)=0  
*Initializes the system before the main solver loop starts.*
- void [add\\_potential](#) ([Potential](#) \*pot)  
*Method for adding a potential to the system.*
- double [get\\_potential\\_energy](#) (const [Walker](#) \*walker)  
*Method for calculating the total potential energy.*
- virtual void [update\\_walker](#) ([Walker](#) \*walker\_pre, const [Walker](#) \*walker\_post, int particle) const =0
- virtual void [reset\\_walker](#) (const [Walker](#) \*walker\_pre, [Walker](#) \*walker\_post, int particle) const =0
- virtual void [calc\\_for\\_newpos](#) (const [Walker](#) \*walker\_old, [Walker](#) \*walker\_new, int particle)=0  
*Method for calculating the necessary values needed by the walker after a new step is made.*
- virtual double [get\\_spatial\\_ratio](#) (const [Walker](#) \*walker\_pre, const [Walker](#) \*walker\_post, int particle)=0  
*Method for calculating the spatial wave function ratios between two subsequent time steps.*
- virtual double [get\\_spatial\\_wf](#) (const [Walker](#) \*walker)=0  
*Method for calculating the spatial wave function's value at a given walkers position.*
- virtual void [get\\_spatial\\_grad](#) ([Walker](#) \*walker, int particle) const =0  
*Method for calculating the changed part of the spatial gradient.*
- virtual void [get\\_spatial\\_grad\\_full](#) ([Walker](#) \*walker) const =0

- Method for calculating the full spatial gradient.*
- virtual double `get_spatial_lapl_sum` (`Walker *walker`) const =0
- Method for calculating the sum of all Laplacians for a given walker.*
- virtual void `copy_walker` (const `Walker *parent`, `Walker *child`) const =0
- Method for copying the system specific parts of a walker.*
- virtual bool `allow_transition` ()=0
- Method allowing the system to override the Metropolis test.*
- void `update_potential_samples` (double weight=1.0)
- void `push_potential_samples` ()
- std::string `dump_samples` (bool mean\_of\_means=false)
- `Orbitals * get_orbital_ptr` ()
- void `set_spin_state` (int `start`, int `end`)

### Protected Attributes

- int `n_p`
- int `n2`
- int `dim`
- int `start`
- The start point of separable calculations.*
- int `end`
- The end point of separable calculations.*
- std::vector< `Potential * > potentials`
- A vector of potentials.*
- `Orbitals * orbital`
- The single particle wave function object.*

### 3.43.1 Detailed Description

The system class separating `Fermions` and `Bosons`. Designed to generalize the solver in terms of particle species.

### 3.43.2 Member Function Documentation

**3.43.2.1** virtual void `System::calc_for_newpos` ( const `Walker * walker_old`, `Walker * walker_new`, int `particle` ) [pure virtual]

Method for calculating the necessary values needed by the walker after a new step is made.

Given a particle number, the method does not recompute unchanged values.

#### Parameters

<code>walker_old</code>	<code>Walker</code> at current time step.
<code>walker_new</code>	<code>Walker</code> at previous time step.

Implemented in `Fermions`, and `Bosons`.

**3.43.2.2** virtual void `System::copy_walker` ( const `Walker * parent`, `Walker * child` ) const [pure virtual]

Method for copying the system specific parts of a walker.



See Also

[QMC::copy\\_walker\(\)](#)

Implemented in [Fermions](#), and [Bosons](#).

#### 3.43.2.3 double System::get\_potential\_energy ( const Walker \* walker )

Method for calculating the total potential energy.

Iterates over all objects in the potentials vector and accumulates their potential energies for the given walker.

#### 3.43.2.4 virtual void System::get\_spatial\_grad ( Walker \* walker, int particle ) const [pure virtual]

Method for calculating the changed part of the spatial gradient.

Depending on which particle we moved, one of the spatial wave function parts (it is split) will be unchanged.

Implemented in [Fermions](#), and [Bosons](#).

#### 3.43.2.5 virtual void System::initialize ( Walker \* walker ) [pure virtual]

Initializes the system before the main solver loop starts.

Called by the [Sampling](#) class when trial positions are set.

Implemented in [Fermions](#), and [Bosons](#).

#### 3.43.2.6 virtual void System::reset\_walker ( const Walker \* walker\_pre, Walker \* walker\_post, int particle ) const [pure virtual]

See Also

[QMC::reset\\_walker\(\)](#)

Implemented in [Fermions](#), and [Bosons](#).

#### 3.43.2.7 void System::set\_spin\_state ( int start, int end ) [inline]

See Also

[QMC::set\\_spin\\_state\(\)](#)

#### 3.43.2.8 virtual void System::update\_walker ( Walker \* walker\_pre, const Walker \* walker\_post, int particle ) const [pure virtual]

See Also

[QMC::update\\_walker\(\)](#)

Implemented in [Fermions](#), and [Bosons](#).

### 3.43.3 Member Data Documentation

#### 3.43.3.1 int System::end [protected]

The end point of separable calculations.

Either  $N/2$  or  $N$ . Since the spatial wave function is split, particles with spin not equal that of the moved particle is unchanged and does not need to be recalculated.

### 3.43.3.2 `int System::start` [protected]

The start point of separable calculations.

Either 0 or  $N/2$ . Since the spatial wave function is split, particles with spin not equal that of the moved particle is unchanged and does not need to be recalculated.

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

- `src/System/System.h`
- `src/System/System.cpp`

## 3.44 SystemObjects Struct Reference

Struct used to initialize system objects.

### Public Attributes

- [Orbitals](#) \* **SP\_basis**
- [Potential](#) \* **onebody\_pot**
- [System](#) \* **SYSTEM**
- [Sampling](#) \* **sample\_method**
- [Jastrow](#) \* **jastrow**

### 3.44.1 Detailed Description

Struct used to initialize system objects.

The memory addresses allocated here will not change throughout the run.

### See Also

[Orbitals](#), [Potential](#), [System](#), [Sampling](#), [Jastrow](#)

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

- `src/structs.h`

## 3.45 VariationalParams Struct Reference

Struct used to initialize the variational parameters.

### Public Attributes

- double [alpha](#)  
*The spatial variational parameter.*
- double [beta](#)  
*The [Jastrow](#) variational parameter.*

### 3.45.1 Detailed Description

Struct used to initialize the variational parameters.

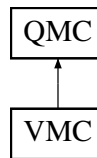
The documentation for this struct was generated from the following file:

- `src/structs.h`

## 3.46 VMC Class Reference

Implementation of the Variational Monte-Carlo Method. Very little needs to be added when the [QMC](#) superclass holds all the general functionality.

Inheritance diagram for VMC:



### Public Member Functions

- [VMC](#) ([GeneralParams](#) &, [VMCparams](#) &, [SystemObjects](#) &, [ParParams](#) &, int [n\\_w](#))  
*Constructor.*
- void **set\_e** (double E)
- double **get\_energy** () const
- void [run\\_method](#) ()  
*Method used for executing the solver.*
- void [output](#) ()  
*Method for standard output.*

### Protected Member Functions

- void [set\\_trial\\_positions](#) ()
- void [store\\_walkers](#) ()  
*Method for storing walkers for [DMC](#).*
- void [save\\_distribution](#) ()  
*Method for storing positional data for the [Distribtuon](#).*
- bool [move\\_authorized](#) (double A)
- void **scale\_values** ()
- void [node\\_comm](#) ()  
*Method for performing node communication.*

### Protected Attributes

- int [pop\\_tresh](#)  
*The amount of cycles between storing walkers for [DMC](#).*
- int [offset](#)  
*The amount of cycles before starting to store walkers for [DMC](#).*
- int [last\\_walker](#)

- Count variable for the last walker stores for [DMC](#).
- double [vmc\\_E](#)  
The [VMC](#) energy.
- [Walker](#) \* [original\\_walker](#)  
The [VMC](#) walker.

## Friends

- class **DMC**
- class **Minimizer**
- class **ASGD**
- class **BlockingData**

### 3.46.1 Detailed Description

Implementation of the Variational Monte-Carlo Method. Very little needs to be added when the [QMC](#) superclass holds all the general functionality.

### 3.46.2 Member Function Documentation

**3.46.2.1** `bool VMC::move_authorized ( double A )` `[inline]`, `[protected]`, `[virtual]`

In [VMC](#), only the metropolis test is performed.

Implements [QMC](#).

**3.46.2.2** `void VMC::save_distribution ( )` `[protected]`, `[virtual]`

Method for storing positional data for the Distribtuon.

Stores the position data of the single [VMC](#) walker every `dist_tresh` cycle after thermalization.

Implements [QMC](#).

**3.46.2.3** `void VMC::set_trial_positions ( )` `[protected]`, `[virtual]`

Sets the trial position for the single walker.

Implements [QMC](#).

**3.46.2.4** `void VMC::store_walkers ( )` `[protected]`

Method for storing walkers for [DMC](#).

Stores the single [VMC](#) walker every `pop_thresh` cycle after offset cycles.

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

- src/QMC/VMC/VMC.h
- src/QMC/VMC/VMC.cpp

## 3.47 VMCparams Struct Reference

Struct used to initialize [VMC](#) parameters.

## Public Attributes

- int `n_c`  
*The number of cycles.*
- double `dt`  
*The time step.*

### 3.47.1 Detailed Description

Struct used to initialize [VMC](#) parameters.

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

- `src/structs.h`

## 3.48 Walker Class Reference

Class representing a Random [Walker](#). Holds position data, alive state, etc. Designed to lighten function arguments, and ease implementation of [QMC](#) methods involving multiple walkers. A lot of values are stored to avoid calculating the same value twice.

## Public Member Functions

- [Walker](#) (int n\_p, int dim, bool alive=true)  
*Constructor.*
- void [calc\\_r\\_i2](#) (int i)  
*Method for calculating the radius squared for one particle.*
- void [calc\\_r\\_i2](#) ()  
*Method for calculating the radius squared for all particles.*
- void [calc\\_r\\_i](#) (int i)  
*Method for calculating the radius of a particle. Assumes the squared exist.*
- void [calc\\_r\\_i](#) ()  
*Method for calculating the radius for all particles.*
- double [calc\\_r\\_rel](#) (int i, int j) const  
*Method for calculating the relative distance between two particles.*
- void [make\\_rel\\_matrix](#) ()
- void [send\\_soul](#) (int dest)
- void [recv\\_soul](#) (int root)
- double [get\\_r\\_i2](#) (int i) const  
*Method for fetching the squared radius of a particle.*
- double [get\\_r\\_i](#) (int i) const  
*Method for calculating the radius of a particle.*
- void [kill](#) ()
- bool [is\\_dead](#) ()
- bool [is\\_alive](#) ()
- void [ressurrect](#) ()
- void [set\\_E](#) (double E)
- double [get\\_E](#) () const
- void [print](#) (std::string header="----") const  
*Prints out all the walkers information.*

## Public Attributes

- double [spatial\\_ratio](#)  
*The ratio of the spatial wave function (stored in the newest walker).*
- double [lapl\\_sum](#)  
*The sum of the Laplacians of all particles.*
- double [E](#)  
*The energy of the given configuration (stored to speed up [DMC](#)).*
- arma::mat [r](#)  
*The positions of all particles.*
- arma::mat [r\\_rel](#)  
*The relative positions of all particles.*
- arma::mat [qforce](#)  
*The Quantum Force for all particles.*
- arma::mat [spatial\\_grad](#)  
*The gradient of the Spatial Wave function for all particles.*
- arma::mat [jast\\_grad](#)  
*The gradient of the [Jastrow](#) Factor for all particles.*
- arma::mat [inv](#)  
*The inverse of the Slater matrix (given fermion system)*
- arma::mat [phi](#)  
*The single particle wave functions for all particles and quantum numbers.*
- arma::field< arma::mat > [dell\\_phi](#)  
*The derivatives of the single particle wave functions for all particles and quantum numbers.*
- arma::cube [dJ](#)  
*Cube used for storing sum terms for the [Jastrow](#) Factor's closed form expressions.*
- arma::rowvec [r2](#)  
*The radius squared for all particles.*
- arma::rowvec [abs\\_r](#)  
*The radius for all particles;.*

## Protected Attributes

- int [n\\_p](#)
- int [n2](#)
- int [dim](#)
- bool [is\\_murdered](#)  
*If true, the walker will be deleted and removed ([DMC](#) only).*

### 3.48.1 Detailed Description

Class representing a Random [Walker](#). Holds position data, alive state, etc. Designed to lighten function arguments, and ease implementation of [QMC](#) methods involving multiple walkers. A lot of values are stored to avoid calculating the same value twice.

### 3.48.2 Constructor & Destructor Documentation

#### 3.48.2.1 Walker::Walker ( int *n\_p*, int *dim*, bool *alive* = true )

Constructor.

Parameters

<i>alive</i>	If false, the walker is initialized dead.
--------------	---

### 3.48.3 Member Function Documentation

#### 3.48.3.1 void Walker::calc\_r\_i( int *i* ) [inline]

Method for calculating the radius of a particle. Assumes the squared exist.

Parameters

<i>i</i>	Particle number.
----------	------------------

#### 3.48.3.2 void Walker::calc\_r\_i2( int *i* )

Method for calculating the radius squared for one particle.

Parameters

<i>i</i>	The particle number.
----------	----------------------

#### 3.48.3.3 double Walker::calc\_r\_rel( int *i*, int *j* ) const

Method for calculating the relative distance between two particles.

Parameters

<i>i,j</i>	The particle numbers.
------------	-----------------------

#### 3.48.3.4 double Walker::get\_r\_i( int *i* ) const [inline]

Method for calculating the radius of a particle.

Parameters

<i>i</i>	Particle number.
----------	------------------

#### 3.48.3.5 double Walker::get\_r\_i2( int *i* ) const [inline]

Method for fetching the squared radius of a particle.

Used in order to avoid calculating the same radius twice.

Parameters

<i>i</i>	Particle number.
----------	------------------

3.48.3.6 `void Walker::kill ( ) [inline]`

Flags the walker for destruction.

See Also

[DMC::bury\\_the\\_dead\(\)](#)

3.48.3.7 `void Walker::make_rel.matrix ( )`

Creates the relative position matrix.

3.48.3.8 `void Walker::print ( std::string header = "----" ) const`

Prints out all the walkers information.

Extremely handy for debugging.

Parameters

<i>header</i>	A header for the printout in order to distinguish several printouts easily.
---------------	---

3.48.3.9 `void Walker::recv_soul ( int root )`

Receives a walker from a different node.

Parameters

<i>root</i>	The rank of the node from which the walker was sent.
-------------	--

3.48.3.10 `void Walker::ressurect ( ) [inline]`

Sets the destruction flag to false.

3.48.3.11 `void Walker::send_soul ( int dest )`

Send a walker to a different node.

Parameters

<i>dest</i>	The receiving node's rank.
-------------	----------------------------

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

- `src/Walker/Walker.h`
- `src/Walker/Walker.cpp`



# Index

ASGD, [9](#)  
    get\_total\_grad, [11](#)  
    get\_variational\_gradients, [11](#)  
AlphaHarmonicOscillator, [7](#)  
    get\_coulomb\_element, [8](#)  
    get\_dell\_alpha\_phi, [8](#)  
    get\_parameter, [8](#)  
    get\_qnums, [8](#)  
    H, [8](#)  
    set\_parameter, [9](#)  
    set\_qnum\_indie\_terms, [9](#)  
AtomCore, [11](#)  
    get\_pot\_E, [12](#)  
  
BasisFunctions, [12](#)  
    eval, [13](#)  
block\_data  
    Blocking, [14](#)  
Blocking, [13](#)  
    block\_data, [14](#)  
    get\_initial\_error, [14](#)  
    get\_unique\_blocks, [14](#)  
Bosons, [14](#)  
    calc\_for\_newpos, [15](#)  
    copy\_walker, [15](#)  
    get\_spatial\_grad, [15](#)  
    get\_spatial\_wf, [15](#)  
    initialize, [15](#)  
    reset\_walker, [16](#)  
    update\_walker, [16](#)  
Brute\_Force, [16](#)  
    copy\_walker, [17](#)  
    get\_necessities, [17](#)  
    reset\_walker, [17](#)  
    update\_walker, [17](#)  
  
calc\_for\_newpos  
    Bosons, [15](#)  
    Fermions, [34](#)  
    System, [76](#)  
calc\_r\_i  
    Walker, [83](#)  
calc\_r\_i2  
    Walker, [83](#)  
calc\_r\_rel  
    Walker, [83](#)  
calculate\_energy\_necessities  
    Importance, [41](#)  
    QMC, [64](#)  
calculate\_local\_energy  
    QMC, [64](#)  
call\_RNG  
    Diffusion, [20](#)  
    Sampling, [69](#)  
combine\_mean  
    ErrorEstimator, [30](#)  
combine\_variance  
    ErrorEstimator, [31](#)  
copy\_walker  
    Bosons, [15](#)  
    Brute\_Force, [17](#)  
    Fermions, [34](#)  
    Importance, [41](#)  
    QMC, [64](#)  
    Sampling, [69](#)  
    System, [76](#)  
Coulomb, [17](#)  
    get\_pot\_E, [18](#)  
cycle  
    QMC, [67](#)  
  
DMC, [24](#)  
    Evolve\_walker, [26](#)  
    iterate\_walker, [26](#)  
    K, [27](#)  
    move\_authorized, [27](#)  
    node\_comm, [27](#)  
    save\_distribution, [27](#)  
    set\_trial\_positions, [27](#)  
    switch\_souls, [27](#)  
DMCparams, [28](#)  
del\_phi  
    DiTransform, [23](#)  
    ExpandedBasis, [32](#)  
    Orbitals, [53](#)  
DiAtomCore, [18](#)  
    get\_pot\_E, [19](#)  
DiTransform, [23](#)  
    del\_phi, [23](#)  
    get\_parameter, [24](#)  
    lapl\_phi, [24](#)  
    phi, [24](#)  
    set\_parameter, [24](#)  
    set\_qnum\_indie\_terms, [24](#)  
diffuse\_walker  
    QMC, [65](#)  
Diffusion, [19](#)  
    call\_RNG, [20](#)  
    get\_g\_ratio, [20](#)  
    get\_new\_pos, [20](#)

- set\_dt, 21
- diffusion
  - Sampling, 70
- Distribution, 21
  - Distribution, 22
  - dump, 22
  - finalize, 22
  - rerun, 22
- DoubleWell, 28
  - get\_pot\_E, 29
- dump
  - Distribution, 22
  - OutputHandler, 56
  - stdoutASGD, 74
  - stdoutDMC, 75
- end
  - Sampling, 70
  - System, 77
- ErrorEstimator, 29
  - combine\_mean, 30
  - combine\_variance, 31
  - ErrorEstimator, 30
  - ErrorEstimator, 30
  - finalize, 31
  - init\_file, 31
  - node\_comm\_gather\_data, 31
  - node\_comm\_scatter\_data, 31
  - update\_data, 31
- estimate\_error
  - QMC, 65
- eval
  - BasisFunctions, 13
- Evolve\_walker
  - DMC, 26
- ExpandedBasis, 32
  - del\_phi, 32
  - lapl\_phi, 32
  - phi, 33
  - set\_qnum\_indie\_terms, 33
- Fermions, 33
  - calc\_for\_newpos, 34
  - copy\_walker, 34
  - get\_spatial\_grad, 34
  - get\_spatial\_wf, 34
  - I, 35
  - initialize, 35
  - make\_merged\_inv, 35
  - reset\_walker, 35
  - update\_walker, 35
- finalize
  - Distribution, 22
  - ErrorEstimator, 31
  - OutputHandler, 56
- Fokker\_Planck, 35
  - get\_g\_ratio, 36
  - get\_new\_pos, 36
- GeneralParams, 37
  - systemConstant, 37
- get\_branching\_Gfunc
  - Sampling, 69
- get\_coulomb\_element
  - AlphaHarmonicOscillator, 8
  - hydrogenicOrbitals, 40
  - Orbitals, 53
- get\_dJ\_matrix
  - Jastrow, 44
  - No\_Jastrow, 49
  - Pade\_Jastrow, 58
- get\_dell\_alpha\_phi
  - AlphaHarmonicOscillator, 8
  - hydrogenicOrbitals, 40
- get\_derivative\_num
  - Jastrow, 44
- get\_g\_ratio
  - Diffusion, 20
  - Fokker\_Planck, 36
  - Sampling, 69
  - Simple, 71
- get\_grad
  - Jastrow, 44
  - No\_Jastrow, 49
  - Pade\_Jastrow, 58
- get\_gradients
  - QMC, 65
- get\_initial\_error
  - Blocking, 14
- get\_j\_ratio
  - Jastrow, 44
  - No\_Jastrow, 49
  - Pade\_Jastrow, 58
- get\_lapl\_sum
  - Jastrow, 45
  - No\_Jastrow, 50
  - Pade\_Jastrow, 58
- get\_laplaciansum\_num
  - Jastrow, 45
- get\_laplsum
  - QMC, 65
- get\_necessities
  - Brute\_Force, 17
  - Importance, 42
  - Sampling, 70
- get\_new\_pos
  - Diffusion, 20
  - Fokker\_Planck, 36
  - Simple, 71
- get\_parameter
  - AlphaHarmonicOscillator, 8
  - DiTransform, 24
  - hydrogenicOrbitals, 40
  - Jastrow, 45
  - No\_Jastrow, 50
  - Orbitals, 53
  - Pade\_Jastrow, 59

- get\_pot\_E
  - AtomCore, 12
  - Coulomb, 18
  - DiAtomCore, 19
  - DoubleWell, 29
  - Harmonic\_osc, 38
  - Potential, 61
- get\_potential\_energy
  - System, 77
- get\_qnums
  - AlphaHarmonicOscillator, 8
- get\_r\_i
  - Walker, 83
- get\_r\_i2
  - Walker, 83
- get\_spatial\_grad
  - Bosons, 15
  - Fermions, 34
  - System, 77
- get\_spatial\_wf
  - Bosons, 15
  - Fermions, 34
- get\_total\_grad
  - ASGD, 11
- get\_unique\_blocks
  - Blocking, 14
- get\_val
  - Jastrow, 45
  - No\_Jastrow, 50
  - Pade\_Jastrow, 59
- get\_variational\_derivative
  - Jastrow, 45
  - No\_Jastrow, 50
  - Orbitals, 53
  - Pade\_Jastrow, 59
- get\_variational\_gradients
  - ASGD, 11
- get\_wf\_value
  - QMC, 65
- H
  - AlphaHarmonicOscillator, 8
- Harmonic\_osc, 38
  - get\_pot\_E, 38
- HartreeFock, 38
- hydrogenicOrbitals, 39
  - get\_coulomb\_element, 40
  - get\_dell\_alpha\_phi, 40
  - get\_parameter, 40
  - set\_parameter, 40
  - set\_qnum\_indie\_terms, 40
- I
  - Fermions, 35
- Importance, 41
  - calculate\_energy\_necessities, 41
  - copy\_walker, 41
  - get\_necessities, 42
  - reset\_walker, 42
  - update\_necessities, 42
  - update\_walker, 42
- init\_file
  - ErrorEstimator, 31
  - OutputHandler, 57
- initialize
  - Bosons, 15
  - Fermions, 35
  - Jastrow, 45
  - No\_Jastrow, 50
  - Pade\_Jastrow, 59
  - System, 77
- iterate\_walker
  - DMC, 26
- Jastrow, 42
  - get\_dJ\_matrix, 44
  - get\_derivative\_num, 44
  - get\_grad, 44
  - get\_j\_ratio, 44
  - get\_lapl\_sum, 45
  - get\_laplaciansum\_num, 45
  - get\_parameter, 45
  - get\_val, 45
  - get\_variational\_derivative, 45
  - initialize, 45
  - set\_parameter, 46
- K
  - DMC, 27
- kill
  - Walker, 83
- lapl\_phi
  - DiTransform, 24
  - ExpandedBasis, 32
  - Orbitals, 54
- make\_merged\_inv
  - Fermions, 35
- make\_rel\_matrix
  - Walker, 84
- metropolis\_test
  - QMC, 65
- Minimizer, 46
  - Minimizer, 47
  - update\_parameters, 47
- MinimizerParams, 47
- move\_authorized
  - DMC, 27
  - QMC, 66
  - VMC, 80
- n\_w
  - QMC, 67
- NO\_STDOUT, 51
- No\_Jastrow, 48
  - get\_dJ\_matrix, 49
  - get\_grad, 49

- get\_j\_ratio, [49](#)
  - get\_lapl\_sum, [50](#)
  - get\_parameter, [50](#)
  - get\_val, [50](#)
  - get\_variational\_derivative, [50](#)
  - initialize, [50](#)
  - set\_parameter, [50](#)
- node\_comm
  - DMC, [27](#)
- node\_comm\_gather\_data
  - ErrorEstimator, [31](#)
- node\_comm\_scatter\_data
  - ErrorEstimator, [31](#)
- num\_ddiff
  - Orbitals, [54](#)
- num\_diff
  - Orbitals, [54](#)
- Orbitals, [51](#)
  - del\_phi, [53](#)
  - get\_coulomb\_element, [53](#)
  - get\_parameter, [53](#)
  - get\_variational\_derivative, [53](#)
  - lapl\_phi, [54](#)
  - num\_ddiff, [54](#)
  - num\_diff, [54](#)
  - phi, [54](#)
  - set\_parameter, [54](#)
  - set\_qnum\_indie\_terms, [54](#)
  - testDell, [55](#)
  - testLaplace, [55](#)
- OutputHandler, [55](#)
  - dump, [56](#)
  - finalize, [56](#)
  - init\_file, [57](#)
  - OutputHandler, [56](#)
  - OutputHandler, [56](#)
- Pade\_Jastrow, [57](#)
  - get\_dJ\_matrix, [58](#)
  - get\_grad, [58](#)
  - get\_j\_ratio, [58](#)
  - get\_lapl\_sum, [58](#)
  - get\_parameter, [59](#)
  - get\_val, [59](#)
  - get\_variational\_derivative, [59](#)
  - initialize, [59](#)
  - set\_parameter, [59](#)
- ParParams, [60](#)
- phi
  - DiTransform, [24](#)
  - ExpandedBasis, [33](#)
  - Orbitals, [54](#)
- Potential, [60](#)
  - get\_pot\_E, [61](#)
- print
  - Walker, [84](#)
- QMC, [61](#)
  - calculate\_energy\_necessities, [64](#)
  - calculate\_local\_energy, [64](#)
  - copy\_walker, [64](#)
  - cycle, [67](#)
  - diffuse\_walker, [65](#)
  - estimate\_error, [65](#)
  - get\_gradients, [65](#)
  - get\_laplsum, [65](#)
  - get\_wf\_value, [65](#)
  - metropolis\_test, [65](#)
  - move\_authorized, [66](#)
  - n\_w, [67](#)
  - QMC, [64](#)
  - QMC, [64](#)
  - reset\_walker, [66](#)
  - save\_distribution, [66](#)
  - set\_spin\_state, [66](#)
  - set\_trial\_positions, [66](#)
  - test\_gradients, [66](#)
  - test\_ratios, [67](#)
  - update\_walker, [67](#)
- recv\_soul
  - Walker, [84](#)
- rerun
  - Distribution, [22](#)
- reset\_walker
  - Bosons, [16](#)
  - Brute\_Force, [17](#)
  - Fermions, [35](#)
  - Importance, [42](#)
  - QMC, [66](#)
  - Sampling, [70](#)
  - System, [77](#)
- ressurrect
  - Walker, [84](#)
- STDOUT, [73](#)
- Sampler, [67](#)
- Sampling, [68](#)
  - call\_RNG, [69](#)
  - copy\_walker, [69](#)
  - diffusion, [70](#)
  - end, [70](#)
  - get\_branching\_Gfunc, [69](#)
  - get\_g\_ratio, [69](#)
  - get\_necessities, [70](#)
  - reset\_walker, [70](#)
  - set\_spin\_state, [70](#)
  - start, [70](#)
  - update\_pos, [70](#)
  - update\_walker, [70](#)
- save\_distribution
  - DMC, [27](#)
  - QMC, [66](#)
  - VMC, [80](#)
- send\_soul
  - Walker, [84](#)
- set\_dt

- Diffusion, [21](#)
- set\_parameter
  - AlphaHarmonicOscillator, [9](#)
  - DiTransform, [24](#)
  - hydrogenicOrbitals, [40](#)
  - Jastrow, [46](#)
  - No\_Jastrow, [50](#)
  - Orbitals, [54](#)
  - Pade\_Jastrow, [59](#)
- set\_qnum\_indie\_terms
  - AlphaHarmonicOscillator, [9](#)
  - DiTransform, [24](#)
  - ExpandedBasis, [33](#)
  - hydrogenicOrbitals, [40](#)
  - Orbitals, [54](#)
- set\_spin\_state
  - QMC, [66](#)
  - Sampling, [70](#)
  - System, [77](#)
- set\_trial\_positions
  - DMC, [27](#)
  - QMC, [66](#)
  - VMC, [80](#)
- Simple, [71](#)
  - get\_g\_ratio, [71](#)
  - get\_new\_pos, [71](#)
- SimpleVar, [72](#)
  - update\_data, [73](#)
- start
  - Sampling, [70](#)
  - System, [78](#)
- stdoutASGD, [73](#)
  - dump, [74](#)
- stdoutDMC, [74](#)
  - dump, [75](#)
- store\_walkers
  - VMC, [80](#)
- switch\_souls
  - DMC, [27](#)
- System, [75](#)
  - calc\_for\_newpos, [76](#)
  - copy\_walker, [76](#)
  - end, [77](#)
  - get\_potential\_energy, [77](#)
  - get\_spatial\_grad, [77](#)
  - initialize, [77](#)
  - reset\_walker, [77](#)
  - set\_spin\_state, [77](#)
  - start, [78](#)
  - update\_walker, [77](#)
- systemConstant
  - GeneralParams, [37](#)
- SystemObjects, [78](#)
- test\_gradients
  - QMC, [66](#)
- test\_ratios
  - QMC, [67](#)
- testDell
  - Orbitals, [55](#)
- testLaplace
  - Orbitals, [55](#)
- update\_data
  - ErrorEstimator, [31](#)
  - SimpleVar, [73](#)
- update\_necessities
  - Importance, [42](#)
- update\_parameters
  - Minimizer, [47](#)
- update\_pos
  - Sampling, [70](#)
- update\_walker
  - Bosons, [16](#)
  - Brute\_Force, [17](#)
  - Fermions, [35](#)
  - Importance, [42](#)
  - QMC, [67](#)
  - Sampling, [70](#)
  - System, [77](#)
- VMC, [79](#)
  - move\_authorized, [80](#)
  - save\_distribution, [80](#)
  - set\_trial\_positions, [80](#)
  - store\_walkers, [80](#)
- VMCparams, [80](#)
- VariationalParams, [78](#)
- Walker, [81](#)
  - calc\_r\_i, [83](#)
  - calc\_r\_i2, [83](#)
  - calc\_r\_rel, [83](#)
  - get\_r\_i, [83](#)
  - get\_r\_i2, [83](#)
  - kill, [83](#)
  - make\_rel\_matrix, [84](#)
  - print, [84](#)
  - recv\_soul, [84](#)
  - ressurrect, [84](#)
  - send\_soul, [84](#)
  - Walker, [83](#)