PX 915: Variational Quantum Monte Carlo - LATIN

GroupB

Generated by Doxygen 1.8.17

Chapter 1

Introduction

This software will compute and visualise the ground state energy of: H, H^{-1}, H_2^+, H_2, He^{+1} and He. This will be done through the use of the variational quantum Monte Carlo (VQMC) method. The main output result will be the minimum energy of the system and the resulting wavefunction.

The expected groundstates are:

H = -13.60 eV ref: `http://hyperphysics.phy-astr.gsu.edu/hbase/hyde.html`

H^{-1} = 14.36 eV ref: `https://journals.aps.org/pra/abstract/10.1103/PhysRevA.43.↩ 6104`

H_2^+ = -16.25 eV ref: `https://doi.org/10.1016/S0009-2614(97)00571-X`

H_2 = -31.72 eV ref: `http://hyperphysics.phy-astr.gsu.edu/hbase/molecule/hmol.↩ html`

He^{+1} = -54.42 eV ref: `http://www.umich.edu/~chem461/QMChap8.pdf`

He = -79.02 eV ref: `http://www.umich.edu/~chem461/QMChap8.pdf`

With the exception of user input and results visualization which are done via Python, the code is written entirely in Fortran. To boost efficiency, parallelization has been implemented through OpenMP with regards to Latin hypercube sampling and Markov chain Monte Carlo integration.

Along with the software documentation, this site provides a Tutorial for installing and running this software and interpreting the results produced. Additionally, the general theory applied in these simulations are explained on the following pages:

- Variational Quantum Monte Carlo
- Monte Carlo Integration
- Latin Hypercube
- Optimisation for a large parameter search-space

1.0.1 Github Repository

The code may be accessed and downloaded from `GitHub`.

1.0.1.1 Fortran Packages Required

- LAPack

1.0.1.2 Python Packages Required

- Numpy
- Matlplotlib
- Tkinter (for the GUI input interface)
- Netcdf

1.0.2 Authors

- Connor Allen
- Adam Fisher
- Peter Lewin-Jones
- Charlotte Rogerson
- Alisdair Soppitt
- Steven Tseng

Chapter 2

Tutorial

This section describes everything needed to set up the code and run the code for simple systems contained in the tutorial.

2.1 Installing/Running the code

This code is compiled and ran in the terminal. YOu can build the code by running the following command for the bash script:

./build_latin_driver.sh

After that has successfully compiled, you can now run the following command code to execute the bash program:

./run_latin_driver.sh

Then the following interface will appear as shown in the figure below. This interface contains all of the variables which the user can change to define the system they want to look at. The interface already contains default values for the variables which the user can keep or change if they wish.

In order to then run the code, the user just needs to change the varibales they wish to change and then click close. This will start running the code with the progress and results displayed in the terminal. When the code has finished running, the output plots will automatically be generated and show each plot in a different pop-up window.

To complete another run of the code, make sure that all the ouputting windows containing the plots are closed, and then re-run the ./run_latin_driver.sh command. There is no need to recompile the code.

In the case that the code fails to run or has crashed after this step, you can easily exit the software by either closing the terminal or using Ctrl+c.

In order to demonstrate the varying functionality of the code, 4 problems have been defined in the subsections below for the user to try. There maybe slight differences in the quoted results due to the sampling and approximate numerical nature of the code, but none of the examples should obtain results which stray far from the results quoted here.

To Improve the accuracy of any the simulations, try increasing the number of trials and the number of steps in the MCMC runs. For 2 electron problems try increasing the number of Jastrow terms from 7 to 9.

2.1.1 Problem 1: Single atom and electron using Slater-type Orbitals

This part of the tutorial contains the most basic system consisting of 1 atom (nuclei) and 1 electron which represents a Hydrogen atom, and is described using Slater-type orbitals. In this example, the Slater-type orbitals provides the analytical result.

All the user has to do is close the user input window as this simulation run uses the default values already provided.

Once the user input window has been closed, the following outputs are printed in the terminal:

- The user defined input values are printed for reference
- Each latin hypercube trial and its corresponding energy result is printed

If the code has ran with no errors or crashes, the following will be printed everytime:

- The best latin hypercube trial
- The minimum energy found in both Hartree and eV units
- The best parameters (this is the optimal degrees of freedom found)
- Comparison of output density statement
- Success in writing output files results.nc4 and xyz.txt
- Total cpu runtime
- Total real runtime
- Upload of result.nc4 successfully
- Print of what is contained in the result.nc4 file
- The number of electrons, atoms

The output plotting script will automatically run, producing the following contour plot of the electron probability density.

The minimum energy obtained for this system is -13.6 eV and -0.499 Hartree energy. The optimal degrees of freedom obtained is -1.42 and 0.997.

The code should take no longer than 3 seconds to run and complete.

2.1.2 Problem 2: Single atom and electron using Gaussian-type Orbitals

This problem is the same set up as the previous problem, however the user instead chooses Gaussian-type orbitals instead of Slater-type orbitals to describe the Hydrogen atom. Also, as the Gaussian-type orbitals only give an approximate result, the total number of steps for MCMC run is increased from 1000000 to 10000000.

This simulation results in the following contour plot as shown in the figure below for the electron probability density.

The energy obtained for this run is -11.8 eV and -0.434 Hartree energy. The energy results for this run may vary as it is an approximate result but the result the user obatins should not stray far from this value due to the increased MCMC steps.

This result is not far from the analytical result

Due to the higher number of MCMC steps defined, this code will take slightly longer than the run completed in Problem 1. The code should not take longer 30 seconds.

2.1.3 Problem 3: Two atoms and a single electron

This problem investigates two atoms (or nuclei) and a single electron system. This corresponds to the H2+ ion.

The output of this system not only produces a contour plot of the electron probability density, but also allows for the evaluation of energy for varying bond lengths.

To first obtain the contour plot, the user uses the same procedure as previously defined in running the scripts, keeping all of the defualt values except the number of atoms is being changed from 1 to 2. Slater-type orbitals are used. The following contour plot should be obtained.

The minimum energy found for this system should be approximately -0.55 Hartree or -15.2 eV.

This code run should take no more than 5 seconds to run.

In order to obtain the energy against varying bond length plot, the user must run the following commands in the terminal:

```
./build_bond_driver.sh
```

Then run

```
./run_bond_driver.sh
```

Which will present the user input interface used previously. The only thing the user needs to change is the number of atoms from 1 to 2. The difference with this run is that the contour plot will not be outputted, only the energy against varying bond length will be plotted. The resulting output plot is shown in the figure below.

In the terminal, when the code has finished running, the minimum energy and its corresponding bond length is outputted. For this example, the bond length with the minimum energy was found to be 2.09 atomic units, with -0.586 Hartree energy.

The bond length driver is currently a proof-of-concept. The parameters for the search over bond length are hard-coded, but can be changed on lines 168-170 of bond_driver.f90 file.

This code run should take no more than 75 seconds to fully run.

If the user wants to revert back to obtaining the contour plots, they will need to repeat the following commands in the terminal:

```
./build_latin_driver.sh

./run_latin_driver.sh
```

2.1.4 Problem 4: Helium+ ion and Helium atom

To simulate a Helium+ ion, 1 atom and 1 electron is simulated but the proton number is changed from 1 to 2, keeping all of the default variables the same for this case.

This results in the following contour plot for the electron density:

The minimum energy obtained is -1.87 Hartree energy or -50.9 eV. This is a good result when compared to expected value of -54.42 eV as the Slater-type orbitals uses hydrogen-like solutions. When using Gaussian-like orbitals, the resulting minimum energy found is -11.6 eV which does not agree well with the expected value.

For the Helium atom, the same procedure is done however the number of electrons is changed from 1 to 2 and the number of protons is changed from 1 to 2 from the default values.

The resulting minimum energy for the Helium atom for Slater-type orbitals was found to be -77.9 eV which is close to the expected value of -79.02 eV. When using Gaussian-like orbtials, the resulting minimum energy was found to be -69.5 eV which is slightly below the expected value, however as it only gives an approximate result, increasing the number of MCMC steps and latin hypercube trials should improve its accuracy.

For the 2 electron case, the contour plot for the resulting electron probability density will be outputted, however the results are wrong and should be ignored.

Using the default number of MCMC steps and latin hypercube trials, the runs should take no more 30 seconds to run each.

2.1.5 User options beyond the tutorial

The user can change the following variables if they want to look at 1 atom only:

- Number of electrons
- Number of atoms
- Number of trials for latin hypercube search
- Number of MCMC steps
- Optional user inputs (either Slater-type or Gaussian-type orbitals)

To investigate 2 atoms, all of the variables defined in the user input interface are relevant.

2.1.6 Plotting Options

These variables allow the user to change aspects of the resulting output such as the domain size and the resolution.

- Amount of distance away from atoms to plot
- Number of points in each direction

Increasing these options will cause the code to take slightly longer to run but will obtain better resolution images as a result.

2.2 Analyzing the results

Results are saved using the subroutine `create_netcdf` and are automatically plotted using either the plotting.py script or the energy_plotting.py script.

Note the user does not actually need to have python or a python IDE open in order for the plotting to appear as it does so automatically through a pop up window. The user also has the option to save the resulting image by clicking the save button with is the button on the far right at the bottom of the window.

2.2.1 Warnings

If the following warning has appeared in the print statements in the terminal: WARNING: Acceptance rate 0 at: 1

The user can ignore this as the code will still run and produce an output. This error relates to an unsuitable initial condition, but unless this error appears later than step 1 then there is no problem, and the MCMC has corrected itself.

2.3 Advanced Options

2.3.1 How to add a basis set

It is relatively easy to add a basis set to the program. Two new functions defining the corresponding single electron wavefunction and reduced hamiltonian must be added to basis_functions.f90, following the format of wave_function_slater_1s() and reduced_hamiltonian_slater_1s(). Pure subfunctions similar to centered_slater_1s_laplacian() can be added to component_functions.f90.

An integer code for the basis must be added to constants.f90. A corresponding check must be added to the case selection on basis_functions::basis_type in the initialisation routine (currently line 198 in basis_functions.f90). This must point the function pointers `wave_function_single` and `reduced_hamiltonian` to the new functions.

2.4 Future extensions

The following additions and extensions may be added to the code.

- Diffusion Monte Carlo
- Other elements

Chapter 3

Variational Quantum Monte Carlo

This page provides a brief overview of the theory behind the variational quantum Monte Carlo method. The mathematics used in the software are then described in more detail such that a user may better understand how the functions and variables utilized within are related.

3.1 VQMC in a Nutshell

The variational quantum Monte Carlo method (VQMC) is a computational method for approximating the ground state energy of a system. More often than not, the time-independent Schrödinger equation of a system defined by the Hamiltonian $\hat{H}$ may not be solved to determine the system's ground state energy E_{gs}. The *variational principle* from quantum mechanics provides that for any normalized function Ψ, we have

$$E_{gs} \leq \langle\Psi|\hat{H}|\Psi\rangle \equiv \langle\hat{H}\rangle, \tag{3.1}$$

i.e., the expected value of $\hat{H}$ with an arbitrary Ψ will most likely overestimate E_{gs}, giving an upper bound for the ground state energy **?**. It follows that E_{gs} may be estimated through testing trial functions across combinations of parameter values and selecting as the approximate ground state wave function the one parameterized with the values that minimize $\langle\hat{H}\rangle$.

The expected value of $\hat{H}$ is given by the following integral

1 $\langle\hat{H}\rangle = \langle\Psi|\hat{H}|\Psi\rangle$
$= \int \Psi^* \hat{H}\Psi d\mathbf{r}_i$,

where $\mathbf{r}_i$ are the positions of the electrons that Ψ depends on. Accordingly, the VQMC method derives its name from using Monte Carlo Integration techniques in evaluating this integral.

3.2 Software Specific Details

This software calculates the ground state energy of the H_2^+ ion and H_2 molecule using VQMC. The Hamiltonian is given by

$$2\hat{H}(\{\mathbf{r}_i\}, \{\mathbf{R}_I\})\Psi(\{\mathbf{r}_i\}) \equiv \left(-\frac{1}{2}\sum_i \nabla_i^2 + \sum_{i>j}\frac{1}{r_{ij}} - \sum_I\sum_i \frac{Z_I}{r_{iI}} + \sum_{I>J}\frac{Z_I Z_J}{r_{IJ}}\right)\Psi = E\Psi, \tag{3.2}$$

where the Born-Oppenheimer approximation is assumed and the units are given in Hartree atomic units. The problem specifies N_n nuclei of atomic number Z_I at coordinates $\mathbf{R}_I$ and N electrons at coordinates $\mathbf{r}_i$ where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$, $r_{iI} = |\mathbf{r}_i - \mathbf{R}_I|$ and $r_{IJ} = |\mathbf{r}_I - \mathbf{r}_J|$.

To apply a Monte Carlo integration method, specifically Markov chain Monte Carlo (MCMC) for this software, the integral given in equation (1) is rewritten as

$$3\langle \hat{H}_L \rangle_{|\Psi|^2} = \frac{\int |\Psi|^2 (\hat{H}\Psi)/\Psi d\{\mathbf{r}_i\}}{\int |\Psi|^2 d\mathbf{r}_i}, \tag{3.3}$$

where $\hat{H}_L = \hat{H}\Psi/\Psi$ is the localized Hamiltonian.

3.2.1 Basis Set Construction

For computational feasibility, trial wave functions parametrized by a reasonably small number of coefficients should be constructed. The number of coefficients also defines the degrees of freedom of the system which is not to be confused with the dimension of the MCMC sample space of $3N$.

The basis set $\{\phi_k\}_{k=1}^{M}$ may be chosen from numerous types of functions. This software utilizes atomic orbitals, e.g., Gaussians, Gaussians with additional properties or Hydrogen-like orbitals, that are centered on $\mathbf{0}$. A set $\{\phi_k^I(\mathbf{r}) = \phi_k(\mathbf{r} - \mathbf{R}_I)\}_{k=1}^{M}$ centered on each atom is then created followed by taking a union over all atoms and renumbering them to get the total basis set:

$$\{\varphi_k(\mathbf{r})\}_{k=1}^{N_n M} = \bigcup_I \{\phi_k^I(\mathbf{r})\}. \tag{3.4}$$

In the software, M is the parameter n_basis_functions_per_atom. For both slater-type basis functions and Gaussians there are 2 parameters per basis function, a linear multiplier and a length scale.

The basis sets in the code have the form, for an atom at the origin, with r the radial distance, α the length scale parameter and c the linear parameter:

Slater-1s:

$$\phi(\mathbf{r}) = c(\alpha^3/\pi)^{1/2} \exp(-\alpha r) \tag{3.5}$$

Sto-3g:

$$\phi(\mathbf{r}) = c(2\alpha/\pi)^{3/4} \exp(-\alpha r^2) \tag{3.6}$$

Single electron wave functions are obtained through summing the basis set in its entirety

$$\psi_i(\mathbf{r}) = \sum_{k=1}^{N_I M} c_{ik} \varphi_k(\mathbf{r}). \tag{3.7}$$

Using this as the trial wave function, i.e., $\Psi = \psi_1$, with MCMC algorithms and optimization schemes results in $2 * N_n \times M$ degrees of freedom. As this is also a linear combination, the linearity of $\hat{H}$ and analytic derivatives of the basis set may be used to speed up the calculation of

$$H\psi_1(\mathbf{r}) = \sum_k c_{ik} H \varphi_k(\mathbf{r}), \tag{3.8}$$

where $H\varphi_k(\mathbf{r})$ can be analytically precomputed for hydrogen-like orbitals or Gaussians.

For systems with more than one electron ($N > 1$), the Slater determinant with Jastrow factor will be taken:

$$\Psi(\{\mathbf{r}_i\}) = e^{J(\{\mathbf{r}_i\})} D(\{\psi_i\}, \{\mathbf{r}_i\}), \tag{3.9}$$

$$D(\{\psi_i\}, \{\mathbf{r}_i\}) = \psi_1(\mathbf{r}_1)\psi_2(\mathbf{r}_1)\cdots\psi_N(\mathbf{r}_1)\psi_1(\mathbf{r}_2)\psi_2(\mathbf{r}_2)\cdots\psi_N(\mathbf{r}_2)\vdots\vdots\ddots\vdots\psi_1(\mathbf{r}_N)\psi_2(\mathbf{r}_N)\cdots\psi_N(\mathbf{r}_N). \tag{3.10}$$

When utilized in full, the number of degrees of freedom (DOF) would be $2 * M \times N_n \times N + N_J$ where N_J is the DOF in J. However, this can be significantly reduced by assuming that a single electron state $\{\psi_i\}$ is sufficient for the calculation, assuming the first 2 electron states differ only by spin: $\psi_1(\mathbf{r}, \uparrow) = \psi_1(\mathbf{r})|\uparrow\rangle$, and $\psi_2 = \psi_1|\downarrow\rangle$. Then this gives for 2 electrons:

$$D(\{\psi_i\}, \{\mathbf{r}_i\}) = \psi_1(\mathbf{r}_1)\psi_2(\mathbf{r}_2) - \psi_1(\mathbf{r}_2)\psi_2(\mathbf{r}_1) = \psi_1(\mathbf{r}_1)\psi_1(\mathbf{r}_2)(|\uparrow\rangle|\downarrow\rangle - |\downarrow\rangle|\uparrow\rangle) = \psi_1(\mathbf{r}_1)\psi_1(\mathbf{r}_2). \tag{3.11}$$

.

This is the expression used in the program and the number of degrees of freedom is only $2 * M \times N_n + N_J$. While this significantly simplifies calculations, it means states where 2 electrons are in different states will not be found.

3.2.2 Jastrow Factor

The choice of Jastrow factor is never trivial. The CASINO code suggests using the DTN factor as one possibility **?**. However, these are designed specifically for periodic systems which is not characteristic of the systems here. Adopting that of Boys and Handy [3,4] would be more suitable:

$$J(\{\mathbf{r}_i\}) = \sum_I \sum_{i<j} U_{Iij}, \tag{3.12}$$

$$U_{Iij} = \sum_k \Delta(m_{kI}, n_{kI}) c_{kI} (\bar{r}_{iI}^{m_{kI}} \bar{r}_{jI}^{n_{kI}} + \bar{r}_{jI}^{m_{kI}} \bar{r}_{iI}^{n_{kI}}) \bar{r}_{ij}^{o_{kI}}, \tag{3.13}$$

$$\bar{r}_{iI} = \frac{b_I r_{iI}}{1 + b_I r_{iI}}, \tag{3.14}$$

$$\bar{r}_{ij} = \frac{d_I r_{ij}}{1 + d_I r_{ij}}, \tag{3.15}$$

$$\Delta(m, n) = 1 - 0.5\delta_{mn}, \tag{3.16}$$

where (m, n, o) is a triple of integers, and including more increases the complexity of J. Furthermore, there are various cusp conditions that must be satisfied **?**, which fixes the choice of these integers. The inverse length scales are assumed to be the same for all atoms, $b_I = b$, $d_I = d$, and we fix them rather than use them as dofs. This is also done with the dofs $c_{kI} = c_k$, so that N_J is the number of terms in the k sum in U_{Iij}.

Chapter 4

Monte Carlo Integration

Monte Carlo (MC) integration is a class of numerical integration techniques having a stochastic-probabilistic nature as opposed to deterministic techniques such as Gaussian quadrature and trapezoidal integration **?**. In general, the MC integration involves taking a simple region that encloses the region of integration, followed by generating random points from the simple region and counting the number of 'hits' or points that are from the integration region . Then in the 2D case, the integral would be evaluated by multiplying the fraction of points that are hits to the total area of the simple region. Thus, as more random points are generated, the integral will converge to the actual value.

This software utilizes the Markov chain Monte Carlo technique (MCMC), specifically Metropolis-Hastings method (see mcmc). Under the Metropolis algorithm, a Markov chain is generated from a proposed density function and moves are accepted/rejected based on a set rule or probability, leading to a probability distribution or final value **?**. If $\mathcal{P}_i^{(n)}$ defines the probability of being in state i at step n then the algorithm is as follows:

- The next state j is sampled with probability $F_{i \to j}$
- State j is accepted with probability $A_{i \to j}$ whereby it is used as the next sample
- State j is rejected with probability $1 - A_{i \to j}$ whereby state i is used as the next sample

With more samples, the properties of F and A become known, leading to $\mathcal{P}_i^{(n \to \infty)} \to p_i$ and thus, the method converges to the true distribution or desired value regardless of the initial state.

Chapter 5

Latin Hypercube

When conducting a search for the optimal parameters, it is easy for the search space dimensions to grow geometrically large. Latin hypercube sampling is a method that creates random sets of parameters that sufficiently cover the search space through

- Subdividing or stratifying each dimension of the search space into N equal regions
- Selecting N points from the search space such that when they're projected onto any dimension, a region in that dimension has only a single point

As explained in **?**, this idea can be applied to a hypercube by first defining a set **X** $= (X_1, X_2, \dots, X_p)$ of p independent random variables. Dividing the domain of each X_j into N intervals allows N samples to be generated and creating N^p intervals. $N+1$ edges then define these intervals

$$\{F_j^{-1}(0), F_j^{-1}(\frac{2}{N}), \dots, F_j^{-1}(\frac{N-1}{N}), F_j^{-1}(N)\}. \tag{5.1}$$

In order to generate a random set of parameters, a permutation matrix Π with size $N \times p$ and elements π_{ij} is defined whereby there are p different columns of randomly selected permutations of the integers $1, 2, \dots, N$. The ith sample of dimension j may then be produced through evaluating

$$x_{ij} = F_j^{-1}(\frac{1}{N}(\pi_{ij} - 1 + u_{ij})), \tag{5.2}$$

where $u_{ij} \sim U(0,1)$, making the ith sample of **X**, $\mathbf{x}_i = (x_{i1}, \dots, x_{iN})$ **?**.

Chapter 6

Optimisation for a large parameter search-space

6.1 Gaussian Process

Given that MCMC evaluations for a given set of points in parameter space can quickly become expensive, we are motivated to use a surrogate model that can be used when searching for the best possible set of parameters to trial. Here we present an approach, as detailed by Wang et al **?**, used to perform a Bayesian global optimisation.

We start by defining our prior distribution on our ground state calculation (as found using MCMC) as a Gaussian process:

$$1f(X) \sim \mathcal{GP}(\mu^{(0)}, \Sigma^{(0)}) \tag{6.1}$$

Where X is of set of points in parameter space, μ is our mean function which is represented as a constant, and Σ defines our covariance matrix which is induced by a Gaussian kernel. The posterior mean and covariance matrix are updated for a set of new points $x^{(1:n)}$ via:

$$2\mu^{(n)} = \mu^{(0)} + K(X, x^{(1:n)})K(x^{(1:n)}, x^{(1:n)})^{-1}(y^{(1:n)} - \mu(x^{(1:n)})) \tag{6.2}$$

$$3\Sigma^{(n)} = K(X, X) - K(X, x^{(1:n)})K(x^{(1:n)}, x^{(1:n)})^{-1}K(x^{(1:n)}, X) \tag{6.3}$$

Where $y^{(1:n)}$ is the value of the function being optimised at a given set of parameters, $\mu(x^{(1:n)})$ is obtained from the prior mean function across the new set, and the function K is computing the covariance (also Gaussian kernel) between each entry in X and $x^{(1:n)}$.

6.2 Optimisation of parameters

Given we have constructed a surrogate model we may evaluate a large set of test points through the expected improvement q-EI:

$$4q\text{-EI} = \mathbb{E}[\max_{i=0,\dots q} e_i[m(X) + C(X)Z]] \tag{6.4}$$

Where e_i is a unit vector which acts as a selector across the other vectors $m(X)$ and $C(X)Z$. The vector $m(X)$ evaluates the difference between the previous best result and the posterior mean, Z is a vector containing samples from a standard normal random vector, and $C(X)$ contains the negative of the Cholesky decomposition of posterior covariance matrix. For the zeroth thread (i=0) both $m(X)$ and $C(X)$ are returned as 0. The stochastic gradient estimator of the expected improvement can then be constructed as:

$$5g(X,Z) = \{ \nabla\ h(X,Z), if \nabla h(X,Z) exists 0, otherwise \tag{6.5}$$

where:

$$6h(X,Z) = \max_{i=0,\ldots q} e_i[m(X) + C(X)Z] \tag{6.6}$$

Following the algorithms outlined by S. P. Smith **?** on the backward differentiation of Cholesky dependent functions one may find the derivatives to $C(X)$ in the above with respect to each parameter. Differentiation over each parameter also needs to be computed for $m(X)$ seperately and added into get the stochastic gradient estimator. From this we may evaluate a gradient estimate at point X_t as:

$$7G(X_t) = \frac{1}{M} \sum_{m=1}^{M} g(X_t, Z_{t,m}) \tag{6.7}$$

Where M is a number of samples the stochastic gradient estimator is averaged over. From this we generate the next set of points in parameter space to test as:

$$8X_{t+1} = H\Pi[X_t + \epsilon_t G(X_t)] \tag{6.8}$$

Where $H\Pi$ defines the projection (back) into the allowed parameter space and ϵ_t defines a step size for the points in parameter to space to wander. The step size decrease as the simulation goes on.

Chapter 7

Acknowledgements

We would like to thank

Dr Peter Brommer, Dr James Kermode, Dr Heather Ratcliffe, Dr Chris Brady, Dr Nicholas Hine

and especially Professor Julie B. Staunton for her weekly advice and guidance.

Chapter 8

Modules Index

8.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 9

Data Type Index

9.1 Data Types List

Here are the data types with brief descriptions:

Chapter 10

File Index

10.1 File List

Here is a list of all files with brief descriptions:

Chapter 11

Module Documentation

11.1 basis_functions Module Reference

Wavefunction, hamiltonian and basis set choice.

Data Types

- interface wave_function_interface

Functions/Subroutines

- subroutine initialise_basis (n_electrons_in, n_basis_functions_per_atom_in, n_atoms_in, atom_coords_in, n_Jastrow_in, fd_length_in, Jastrow_b_length_in, Jastrow_d_length_in, proton_numbers_in, basis_type_in)

  Initialisation routine.
- subroutine deinitialise_basis

  Dinitialisation routine.
- real(dp) function wave_function_slater_1s (position, dof_coefficients)

  Single Electron wavefunction: slater 1s basis.
- real(dp) function reduced_hamiltonian_slater_1s (position, dof_coefficients)

  Single Electron Reduced Hamiltonian: slater 1s basis.
- real(dp) function discrete_laplacian_reduced (position, h, dofs)

  Finite Difference Reduced Laplacian Computes finite difference approximation to the reduced laplacian $(\nabla^2\psi)/\psi$.
- real(dp) function wave_function_2_electrons (position, dof_coefficients)

  Wavefunction for 2 electrons.
- real(dp) function reduced_hamiltonian_2_electrons (position, dof_coefficients)

  Reduced Hamiltonian for 2 electrons.
- subroutine mno_allocate (n_terms)

  Allocate parameters for Jastrow factor.
- real(dp) function log_density (position, dof_coefficients)

  Log of the Probability Density.
- real(dp) function wave_function_sto3g (position, dof_coefficients)

  Single Electron wavefunction: gaussian sto3g basis.
- real(dp) function reduced_hamiltonian_sto3g (position, dof_coefficients)

  Single Electron Reduced Hamiltonian: gaussian sto3g basis.

Variables

- procedure(wave_function_interface), pointer wave_function
- procedure(wave_function_interface), pointer reduced_hamiltonian
- logical, protected initialised = .false.
- integer, protected n_electrons
- integer, protected n_basis_functions_per_atom
- integer, protected n_atoms
- real(dp), dimension(:,:), allocatable, protected atom_coords
- integer, protected n_jastrow_dofs
- real(dp), protected fd_h
- real(dp), protected b_length
- real(dp), protected d_length
- real(dp), dimension(:), allocatable, protected proton_numbers
- integer, protected basis_type
- integer, protected number_dofs
- integer, protected n_space_dims
- real(dp), dimension(:,:), allocatable, protected dof_bounds
- integer, protected n_dofs_per_atom
- integer, protected n_dofs_no_jastrow
- integer, dimension(:,:), allocatable, protected mno_parameters
- procedure(wave_function_interface), pointer wave_function_single

11.1.1 Detailed Description

Wavefunction, hamiltonian and basis set choice.

This module contains functions that define the hamiltonian that specifies the problem and the basis set being used. Makes significant use of module variables defined in an initialisation subroutine. This allocates 2 function pointers using wave_function_interface

- wave_function
- reduced_hamiltonian

Parameters

position	a real(dp) array of electron coordinates
dof_coefficients	a real(dp) array of degree of freedom (dof) parameters These are the only functions that should be used outside of the module.

11.1.2 Function/Subroutine Documentation

11.1.2.1 deinitialise_basis()

```
subroutine basis_functions::deinitialise_basis
```

Dinitialisation routine.

MUST be run at the end of main program to deallocate module variables.

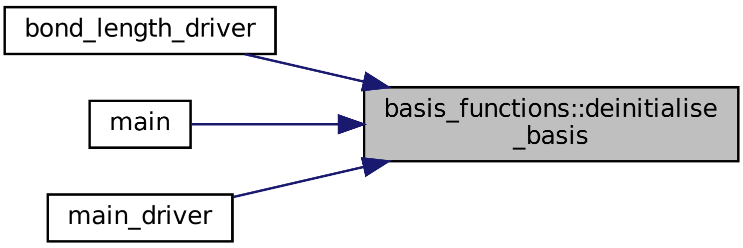
Definition at line 267 of file basis_functions.f90.

```
    implicit none
    if (initialised) deallocate(dof_bounds,atom_coords,proton_numbers)
    if (allocated(mno_parameters)) deallocate(mno_parameters)
    initialised = .false.
```

References atom_coords, dof_bounds, initialised, mno_parameters, and proton_numbers.

Referenced by bond_length_driver(), main(), and main_driver().

Here is the caller graph for this function:



11.1.2.2 discrete_laplacian_reduced()

```
real(dp) function basis_functions::discrete_laplacian_reduced (
             real(dp), dimension(:), intent(in) position,
             real(dp), intent(in) h,
             real(dp), dimension(:), intent(in) dofs )
```

Finite Difference Reduced Laplacian Computes finite difference approximation to the reduced laplacian $(\nabla^2\psi)/\psi$.

Uses that $f''(x)/f(x) =((f(x+h)+f(x-h))/f(x)-2)/(h^2)+O(h^2)$ (∗)

Parameters

in	*position*	Space coordinate of electrons
in	*h*	Finite difference lengthscale
in	*dofs*	Values of the dofs

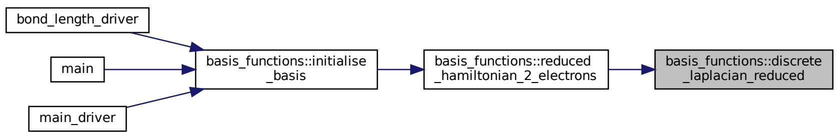
Definition at line 368 of file basis_functions.f90.

```

    implicit none
    real(dp) :: discrete_Laplacian_reduced
    real(dp), dimension(:), intent(in) :: position
    real(dp), intent(in) :: h
    real(dp), dimension(:), intent(in) :: dofs
    integer :: i ! Loop variable
    real(dp), dimension(:,:), allocatable :: delta ! temporary array for vector displacements

    ! Initialise and bounds tests
    if (.not.(initialised)) then
      print *, "Error, basis not initialised"
      stop
    end if
    if (size(position).ne.n_space_dims) then
      print *, "Error, wrong space dimension, got", size(position), "wanted", n_space_dims
      stop
    end if
    if (size(dofs).ne.number_dofs) then
      print *, "Error, wrong number of dofs, got", size(dofs), "wanted", number_dofs
      stop
    end if

    ! Create matrix of cartesian basis vectors of length h. Set as 0 initially, h added during main loop
    allocate(delta(n_space_dims,n_space_dims))
    delta = 0.0_dp ! Matrix assignment
    discrete_laplacian_reduced = 0.0_dp
    do i = 1, n_space_dims ! Loop over space dimensions
      delta(i,i) = h
      discrete_laplacian_reduced = discrete_laplacian_reduced &
      ! Following (*), sum of (f(x+h)+f(x-h)) in all space dimensions
      + wave_function(position + delta(:,i), dofs) + wave_function(position - delta(:,i), dofs)
    end do
    ! Following (*), divide by $f(x)$, and subtract 2 for each space dimension
    discrete_laplacian_reduced = discrete_laplacian_reduced/wave_function(position ,dofs) -
   2*n_space_dims
    ! Following (*), divide by h^2
    discrete_laplacian_reduced = discrete_laplacian_reduced/(h**2)

    ! deallocate temporary array
    deallocate(delta)
```

References initialised, n_space_dims, number_dofs, and wave_function.

Referenced by reduced_hamiltonian_2_electrons().

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::reduced_hamiltonian_2_electrons
basis_functions::discrete_laplacian_reduced

11.1.2.3 initialise_basis()

```
subroutine basis_functions::initialise_basis (
              integer, intent(in) n_electrons_in,
              integer, intent(in) n_basis_functions_per_atom_in,
              integer, intent(in) n_atoms_in,
              real(dp), dimension(:,:), intent(in) atom_coords_in,
              integer, intent(in), optional n_Jastrow_in,
              real(dp), intent(in), optional fd_length_in,
              real(dp), intent(in), optional Jastrow_b_length_in,
              real(dp), intent(in), optional Jastrow_d_length_in,
              real(dp), dimension(:), intent(in), optional proton_numbers_in,
              integer, intent(in), optional basis_type_in )
```

Initialisation routine.

Must be run at the start of the main program, after user input. This defines the problem and selects the basis being used. It allocates module variables and procedure pointers.

Parameters

in	*n_electrons_in*	Number of electrons
in	*n_basis_functions_per_atom←_in*	Number of linear terms in the single-electron wavefunction per atom
in	*n_atoms_in*	Number of atoms
in	*atom_coords_in*	Coordinates of the atoms. Shape (3,n_atoms)
in	*n_jastrow_in*	Number of dofs in the Jastrow (interaction) term
in	*fd_length_in*	Lengthscale of the finite difference code
in	*jastrow_b_length_in*	Inverse lengthscale of nuclear-electron interaction
in	*jastrow_d_length_in*	Inverse lengthscale of electron-electron interaction
in	*proton_numbers_in*	Proton Numbers of atoms
in	*basis_type_in*	integer code for type of basis. Codes are listed in shared_constants.f90

Definition at line 69 of file basis_functions.f90.

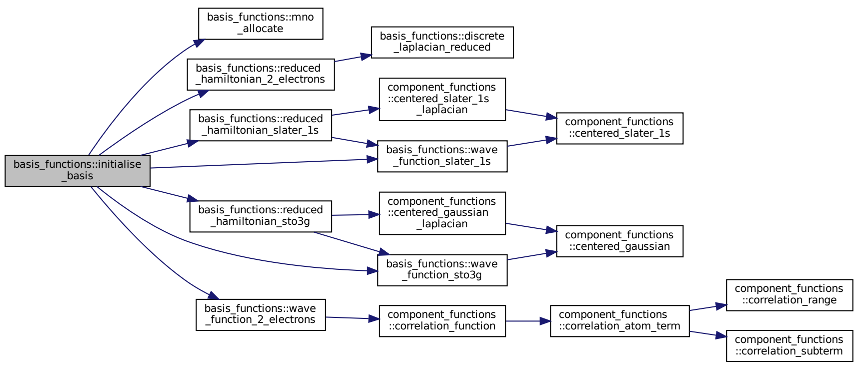
```
implicit none
! Required Parameters
integer, intent(in) :: n_electrons_in
integer, intent(in) :: n_basis_functions_per_atom_in
integer, intent(in) :: n_atoms_in
real(dp), dimension(:,:), intent(in) :: atom_coords_in
! Optional Paramters
integer, optional, intent(in) :: n_Jastrow_in
real(dp), optional, intent(in) :: fd_length_in
real(dp), optional, intent(in) :: Jastrow_b_length_in
real(dp), optional, intent(in) :: Jastrow_d_length_in
real(dp), dimension(:), optional, intent(in) :: proton_numbers_in
integer, optional, intent(in) :: basis_type_in
! Internal Variables
integer :: i ! Loop variable
integer :: n_dofs_per_basis_function ! number of dofs per linear term of the wavefunction. Always 2
for slater and Gaussian

! Check not already initialised
if (initialised) then
  print *, "Error, cannot initialise basis twice"
  stop
 end if

 if (n_atoms_in <= 0) then
   print*,"Error, number of atoms must be positive"
   stop
 end if
 if ((n_atoms_in .ne. 1).and.(n_atoms_in .ne. 2)) then
   print*,"Error, only 1 or 2 atoms supported"
   stop
 end if
 n_atoms = n_atoms_in

 ! Check and allocate the atom coordinates
 if (size(atom_coords_in,2) .ne. n_atoms_in ) then
   print*,"Error, wrong number of atoms"
   stop
 end if
 if (size(atom_coords_in,1) .ne. 3 ) then
   print*,"Error, wrong dimension of atom coordinates"
   stop
 end if
 allocate(atom_coords(3,size(atom_coords_in,2)))
 atom_coords = atom_coords_in

 ! Check and assign other required inputs
 if (n_basis_functions_per_atom_in <= 0) then
   print*,"Error, number of basis functions per atom must be positive"
```

```
      stop
    end if
    n_basis_functions_per_atom = n_basis_functions_per_atom_in

    if ((n_electrons_in .ne. 1).and.(n_electrons_in .ne. 2)) then
      print*,"Error, number of electrons must be 1 or 2"
      stop
    end if
    n_electrons = n_electrons_in

    n_space_dims = 3*n_electrons



    ! Check and assign optional paramters
    if (present(n_jastrow_in)) then
      n_jastrow_dofs = n_jastrow_in
    else
      n_jastrow_dofs = 7 ! Default Value
    end if

    if (present(fd_length_in)) then
      if (fd_length_in>0.00009) then
        fd_h = fd_length_in
      else
        print*, "Error, finite difference length must be at least 0.0001"
        stop
      end if
    else
      fd_h = 0.01_dp ! Default Value
    end if

    if (present(jastrow_b_length_in)) then
      if (jastrow_b_length_in>0.05) then
        b_length = jastrow_b_length_in
      else
        print*, "Error, Jastrow b length must be positive"
        stop
      end if
    else
      b_length = 1.0_dp ! Default value
    end if

    if (present(jastrow_d_length_in)) then
      if (jastrow_d_length_in>0.05) then
        d_length = jastrow_d_length_in
      else
        print*, "Error, Jastrow d length must be positive"
        stop
      end if
    else
      d_length = 1.0_dp ! Default value
    end if
    allocate(proton_numbers(n_atoms))
    if (present(proton_numbers_in)) then
      if (size(proton_numbers_in)==n_atoms) then
        proton_numbers = proton_numbers_in
      else
        print*, "Error, mismatch in number of atoms and proton numbers"
        stop
      end if
    else
      proton_numbers = 1.0_dp ! Default value is hydrogen
    end if

    if(present(basis_type_in)) then
      basis_type = basis_type_in
    else
      basis_type = slater_1s_code !Default basis set
    end if

    ! Assign individual electron wavefunction with input basis set
    select case(basis_type)
    case (slater_1s_code)
      wave_function_single => wave_function_slater_1s
      reduced_hamiltonian => reduced_hamiltonian_slater_1s
    case (sto_3g_code)
      wave_function_single => wave_function_sto3g
      reduced_hamiltonian => reduced_hamiltonian_sto3g
    case default
      print *, "Basis type not recognised"
      stop
    end select

    ! Set number of dofs per basis function. This may change for future choices of basis sets
    n_dofs_per_basis_function = 2
    n_dofs_per_atom = n_dofs_per_basis_function * n_basis_functions_per_atom
    n_dofs_no_jastrow = n_dofs_per_atom * n_atoms

    ! Assignments for one electron, slater type orbital
    if (n_electrons_in .eq. 1) then

      number_dofs = n_dofs_per_atom * n_atoms

      allocate(dof_bounds(number_dofs,2))

      ! Assign dof bounds for single electron part
      ! Ordering here is for 2 dofs per basis function, in pairs, first linear dof then width dof
      do i = 1, n_basis_functions_per_atom*n_atoms ! Loop over pairs of dofs
          dof_bounds(2*i-1,:)=[-1.5_dp,1.5_dp] ! Linear coeffs defaults -1.5 to 1.5
          dof_bounds(2*i,:)=[0.1_dp,1.5_dp]    ! width positive, up to 1.5
      end do

      ! Assign wave_function pointer. Ham pointer already assigned
      wave_function => wave_function_single
    end if

    ! Assignments for 2 electrons, slater type orbital
    if (n_electrons_in .eq. 2) then

      ! Allocate the mno parameters that determine the form of Jastrow term
      call mno_allocate(n_jastrow_dofs)

      number_dofs = n_dofs_per_atom * n_atoms + n_jastrow_dofs

      allocate(dof_bounds(number_dofs,2))

      ! Assign dof bounds
      ! Ordering here is for 2 dofs per basis function, first linear cooef then width coeff
      do i = 1,n_basis_functions_per_atom*n_atoms ! Loop over pairs of dofs
          dof_bounds(2*i-1,:)=[-1.5_dp,1.5_dp] ! Linear coeffs bounds -1.5 to 1.5
          dof_bounds(2*i,:)=[0.1_dp,1.5_dp]    ! width positive, up to 1.5
      end do
      if (n_jastrow_dofs.ne.0) then
        do i = n_dofs_per_atom*n_atoms+1, number_dofs
          dof_bounds(i,:) = [-1.5_dp,1.5_dp] ! Jastrow dofs bounds -1.5 to 1.5
        end do
      end if
      ! Assign the procedure pointers
      wave_function => wave_function_2_electrons
      reduced_hamiltonian => reduced_hamiltonian_2_electrons
    end if

    ! Set initialised flag, which is tested in the routines
    initialised = .true.
```

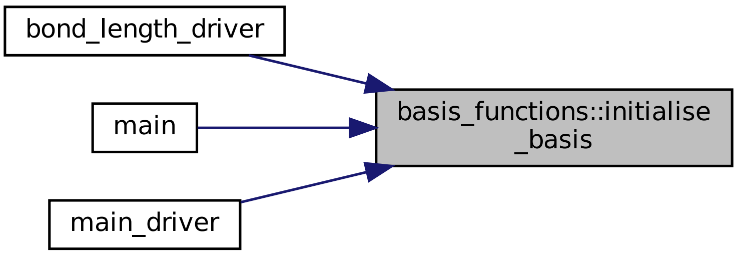
References atom_coords, b_length, basis_type, d_length, dof_bounds, fd_h, initialised, mno_allocate(), n_atoms, n_basis_functions_per_atom, n_dofs_no_jastrow, n_dofs_per_atom, n_electrons, n_jastrow_dofs, n_space_dims, number_dofs, proton_numbers, reduced_hamiltonian, reduced_hamiltonian_2_electrons(), reduced_hamiltonian←_slater_1s(), reduced_hamiltonian_sto3g(), shared_constants::slater_1s_code, shared_constants::sto_3g_code, wave_function, wave_function_2_electrons(), wave_function_single, wave_function_slater_1s(), and wave_←function_sto3g().

Referenced by bond_length_driver(), main(), and main_driver().

Here is the call graph for this function:


basis_functions::initialise_basis
basis_functions::mno_allocate
basis_functions::reduced_hamiltonian_2_electrons
basis_functions::discrete_laplacian_reduced
basis_functions::reduced_hamiltonian_slater_1s
component_functions::centered_slater_1s_laplacian
basis_functions::wave_function_slater_1s
component_functions::centered_slater_1s
basis_functions::reduced_hamiltonian_sto3g
component_functions::centered_gaussian_laplacian
component_functions::centered_gaussian
basis_functions::wave_function_sto3g
basis_functions::wave_function_2_electrons
component_functions::correlation_function
component_functions::correlation_atom_term
component_functions::correlation_range
component_functions::correlation_subterm

Here is the caller graph for this function:



11.1.2.4 log_density()

```
real(dp) function basis_functions::log_density (
             real(dp), dimension(:), intent(in) position,
             real(dp), dimension(:), intent(in) dof_coefficients )
```

Log of the Probability Density.

For the MCMC integration.

Parameters

in	*position*	Space coordinate of electrons
in	*dof_coefficients*	Values of the dofs

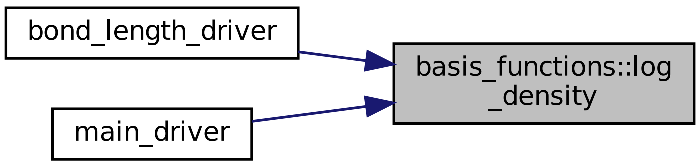
Definition at line 540 of file basis_functions.f90.

```
    implicit none
    real(dp) :: log_density
    real(dp), dimension(:), intent(in) :: position
    real(dp), dimension(:), intent(in) :: dof_coefficients

    log_density = 2*log(abs(wave_function(position,dof_coefficients)))
```

References wave_function.

Referenced by bond_length_driver(), and main_driver().

Here is the caller graph for this function:



11.1.2.5 mno_allocate()

```
subroutine basis_functions::mno_allocate (
             integer, intent(in) n_terms )
```

Allocate parameters for Jastrow factor.

Allocates the paramters that determine type of Jastrow function. Sizes 0,3,7,9 supported. Functions are from Schmidt and Moskowitz 1990

Parameters

in	*n_terms*	Number of terms in Jastrow correlation function

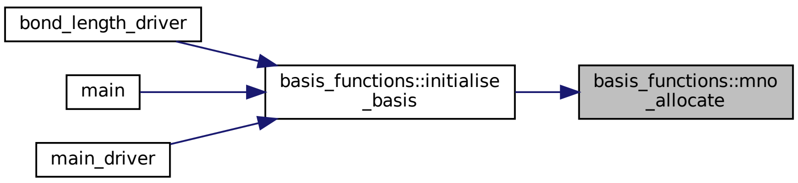
Definition at line 497 of file basis_functions.f90.

```
    implicit none
    integer,intent(in) :: n_terms
    select case (n_terms)
      case (0)
        n_jastrow_dofs = 0
      case (3)
        allocate(mno_parameters(3,3))
        mno_parameters(:,1) = [0,0,1]
        mno_parameters(:,2) = [0,0,2]
        mno_parameters(:,3) = [2,0,0]
        n_jastrow_dofs = 3
      case (7)
        allocate(mno_parameters(3,7))
        mno_parameters(:,1) = [0,0,1]
        mno_parameters(:,2) = [0,0,2]
        mno_parameters(:,3) = [0,0,3]
        mno_parameters(:,4) = [0,0,4]
        mno_parameters(:,5) = [2,0,0]
        mno_parameters(:,6) = [3,0,0]
        mno_parameters(:,7) = [4,0,0]
        n_jastrow_dofs = 7
      case (9)
        allocate(mno_parameters(3,9))
        mno_parameters(:,1) = [0,0,1]
        mno_parameters(:,2) = [0,0,2]
        mno_parameters(:,3) = [0,0,3]
        mno_parameters(:,4) = [0,0,4]
        mno_parameters(:,5) = [2,0,0]
        mno_parameters(:,6) = [3,0,0]
        mno_parameters(:,7) = [4,0,0]
        mno_parameters(:,8) = [2,2,0]
        mno_parameters(:,9) = [2,0,2]
        n_jastrow_dofs = 9
      case default
        print *, "Jastrow term of size:", n_terms, "not supported"
        stop
    end select
```

References mno_parameters, and n_jastrow_dofs.

Referenced by initialise_basis().

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise _basis
basis_functions::mno _allocate

11.1.2.6 reduced_hamiltonian_2_electrons()

```
real(dp) function basis_functions::reduced_hamiltonian_2_electrons (
            real(dp), dimension(:), intent(in) position,
            real(dp), dimension(:), intent(in) dof_coefficients )
```

Reduced Hamiltonian for 2 electrons.

Computes the reduced Hamiltonian $(H\psi)/\psi$ for 2 electrons. Uses discrete_Laplacian_reduced in this module.

Parameters

in	*position*	Space coordinate of electrons
in	*dof_coefficients*	Values of the dofs

Definition at line 454 of file basis_functions.f90.

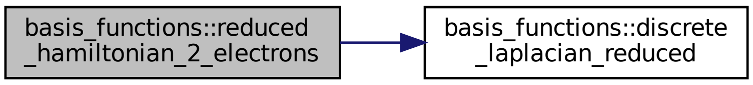
```
  implicit none
  real(dp) :: reduced_hamiltonian_2_electrons
  real(dp), dimension(:), intent(in) :: position
  real(dp), dimension(:), intent(in) :: dof_coefficients
  integer :: j ! Loop variable
  ! Initialise and bounds tests
  if (.not.(initialised)) then
    print *, "Error, basis not initialised"
    stop
  end if
  if (size(position).ne.n_space_dims) then
    print *, "Error, wrong space dimension, got", size(position), "wanted", n_space_dims
    stop
  end if
  if (size(dof_coefficients).ne.number_dofs) then
    print *, "Error, wrong number of dofs, got", size(dof_coefficients), "wanted", number_dofs
    stop
  end if

  ! Reduced Kinetic energy and electron-electron reduced potential energy
  ! $-0.5\nabla^2\psi$ +1/r, r distance between electrons
```

```
    reduced_hamiltonian_2_electrons = -0.5_dp*discrete_laplacian_reduced(position, fd_h,
   dof_coefficients) &
      + 1.0_dp /norm2(position(1:3)-position(4:6))

    ! Reduced electron-nuclear potential energy, -1/r, r distance from electron to atom
    do j = 1, n_atoms ! Loop over atoms
      reduced_hamiltonian_2_electrons  = reduced_hamiltonian_2_electrons  &
          -proton_numbers(j) /norm2(position(1:3)-atom_coords(:,j))&
          -proton_numbers(j) /norm2(position(4:6)-atom_coords(:,j))
    end do

    if (n_atoms ==2) then
      reduced_hamiltonian_2_electrons  = reduced_hamiltonian_2_electrons  &
      +proton_numbers(1)*proton_numbers(2) /norm2(atom_coords(:,1)-atom_coords(:,2))
    end if
```

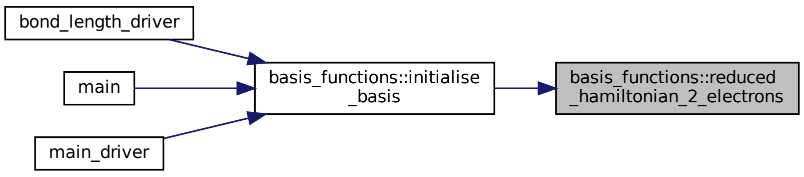
References atom_coords, discrete_laplacian_reduced(), fd_h, initialised, n_atoms, n_space_dims, number_dofs, and proton_numbers.

Referenced by initialise_basis().

Here is the call graph for this function:



Here is the caller graph for this function:



11.1.2.7 reduced_hamiltonian_slater_1s()

```
real(dp) function basis_functions::reduced_hamiltonian_slater_1s (
             real(dp), dimension(:), intent(in) position,
             real(dp), dimension(:), intent(in) dof_coefficients )
```

Single Electron Reduced Hamiltonian: slater 1s basis.

Reduced Hamiltonian $(H\psi)/\psi$ for 1 electron, with slater 1s type orbital Uses centered_slater_1s_laplacian in component_functions.f90 Used only for 1 electron problems.

Parameters

in	*position*	Space coordinate of electrons
in	*dof_coefficients*	Values of the dofs

loop over atoms

loop over linear terms in wavefunction

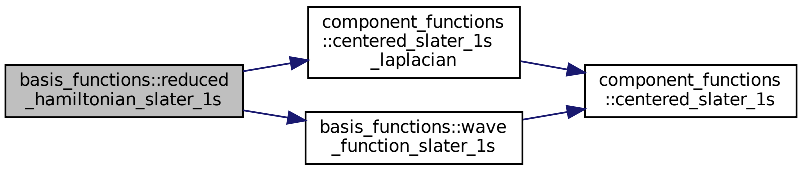
Definition at line 315 of file basis_functions.f90.

```
    implicit none
    real(dp) :: reduced_hamiltonian_slater_1s
    real(dp), dimension(:), intent(in) :: position
    real(dp), dimension(:), intent(in) :: dof_coefficients
    integer :: i, j ! Loop variables

    ! Initialise and bounds tests
    if (.not.(initialised)) then
      print *, "Error, basis not initialised"
      stop
    end if
    if (size(position).ne.n_space_dims) then
      print *, "Error, wrong space dimension, got", size(position), "wanted", n_space_dims
      stop
    end if
    if (size(dof_coefficients).ne.number_dofs) then
      print *, "Error, wrong number of dofs, got", size(dof_coefficients), "wanted", number_dofs
      stop
    end if

    reduced_hamiltonian_slater_1s = 0.0_dp
    ! Kinetic energy term $-0.5\nabla^2\psi$
    ! This wavefunction is linear, and it is easy to compute this analytically
    do j = 0, n_atoms-1
      do i= 1,n_basis_functions_per_atom
        reduced_hamiltonian_slater_1s = reduced_hamiltonian_slater_1s &
          -0.5_dp * dof_coefficients( 2*i-1 +j*n_dofs_per_atom) & ! linear coefficient dof
            *centered_slater_1s_laplacian(position-atom_coords(:,j+1), & ! position relative to atom
              dof_coefficients( 2*i +j*n_dofs_per_atom ) ) ! lengthscale dof
      end do
    end do
    ! Divide by the wavefunction to get reduced laplacian $(-0.5\nabla^2\psi)/\psi$
    reduced_hamiltonian_slater_1s =
   reduced_hamiltonian_slater_1s/wave_function_slater_1s(position,dof_coefficients)

    ! Potential energy, already reduced, so no wavefunction in this
    do j = 1, n_atoms ! loop over atoms
    reduced_hamiltonian_slater_1s = reduced_hamiltonian_slater_1s &
      -proton_numbers(j) /norm2(position-atom_coords(:,j)) ! -1/r, r distance from electron to atom
    end do

    if (n_atoms ==2) then
      reduced_hamiltonian_slater_1s = reduced_hamiltonian_slater_1s &
      +proton_numbers(1)*proton_numbers(2) /norm2(atom_coords(:,1)-atom_coords(:,2))
    end if

```

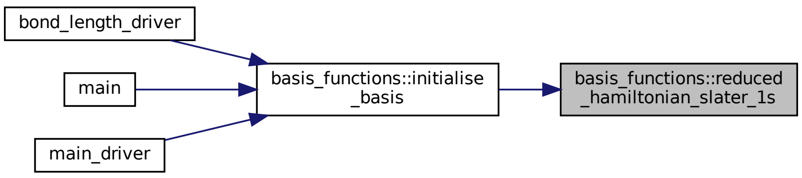
References atom_coords, component_functions::centered_slater_1s_laplacian(), initialised, n_atoms, n_basis↩_functions_per_atom, n_dofs_per_atom, n_space_dims, number_dofs, proton_numbers, and wave_function_↩slater_1s().

Referenced by initialise_basis().

Here is the call graph for this function:


basis_functions::reduced_hamiltonian_slater_1s
component_functions::centered_slater_1s_laplacian
basis_functions::wave_function_slater_1s
component_functions::centered_slater_1s

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::reduced_hamiltonian_slater_1s

11.1.2.8 reduced_hamiltonian_sto3g()

```
real(dp) function basis_functions::reduced_hamiltonian_sto3g (
            real(dp), dimension(:), intent(in) position,
            real(dp), dimension(:), intent(in) dof_coefficients )
```

Single Electron Reduced Hamiltonian: gaussian sto3g basis.

Reduced Hamiltonian $(H\psi)/\psi$ for 1 electron, with basic gaussian sto3g type orbital Uses centered_gaussian←_laplacian in component_functions.f90 Used only for 1 electron problems.

Parameters

in	*position*	Space coordinate of electrons
in	*dof_coefficients*	Values of the dofs

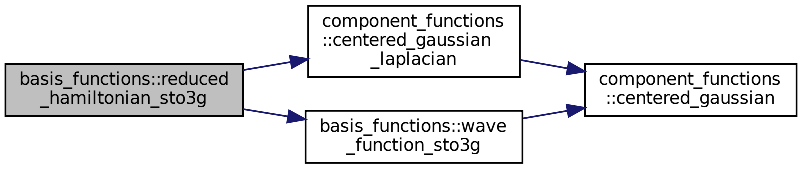
Definition at line 594 of file basis_functions.f90.

```
    implicit none
    real(dp) :: reduced_hamiltonian_sto3g
    real(dp), dimension(:), intent(in) :: position
    real(dp), dimension(:), intent(in) :: dof_coefficients
    integer :: i, j ! Loop variables

    ! Initialise and bounds tests
    if (.not.(initialised)) then
      print *, "Error, basis not initialised"
      stop
    end if
    if (size(position).ne.n_space_dims) then
      print *, "Error, wrong space dimension, got", size(position), "wanted", n_space_dims
      stop
    end if
    if (size(dof_coefficients).ne.number_dofs) then
      print *, "Error, wrong number of dofs, got", size(dof_coefficients), "wanted", number_dofs
      stop
    end if

    reduced_hamiltonian_sto3g = 0.0_dp
    ! Kinetic energy term $-0.5\nabla^2\psi$
    ! This wavefunction is linear, and it is easy to compute this analytically
    do j = 0, n_atoms-1 ! loop over atoms
      do i= 1,n_basis_functions_per_atom ! loop over linear terms in wavefunction
        reduced_hamiltonian_sto3g = reduced_hamiltonian_sto3g &
          -0.5_dp * dof_coefficients( 2*i-1 +j*n_dofs_per_atom) & ! linear coefficient dof
            *centered_gaussian_laplacian(position-atom_coords(:,j+1), & ! position relative to atom
              dof_coefficients( 2*i +j*n_dofs_per_atom ) ) ! lengthscale dof
      end do
    end do
    ! Divide by the wavefunction to get reduced laplacian $(-0.5\nabla^2\psi)/\psi$
    reduced_hamiltonian_sto3g = reduced_hamiltonian_sto3g/wave_function_sto3g(position,dof_coefficients)

    ! Potential energy, already reduced, so no wavefunction in this
    do j = 0, n_atoms-1 ! loop over atoms
      reduced_hamiltonian_sto3g = reduced_hamiltonian_sto3g &
      -1.0_dp /norm2(position-atom_coords(:,j+1)) !-1/r, r distance from electron to atom
    end do

    if (n_atoms ==2) then
      reduced_hamiltonian_sto3g = reduced_hamiltonian_sto3g &
      +proton_numbers(1)*proton_numbers(2) /norm2(atom_coords(:,1)-atom_coords(:,2))
    end if

```

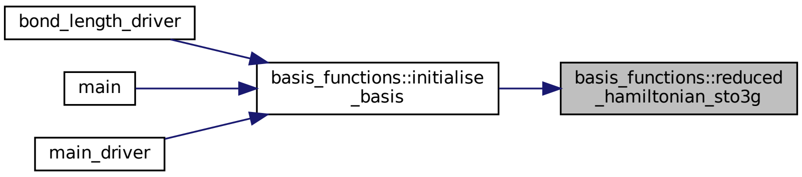
References atom_coords, component_functions::centered_gaussian_laplacian(), initialised, n_atoms, n_basis←_functions_per_atom, n_dofs_per_atom, n_space_dims, number_dofs, proton_numbers, and wave_function_←sto3g().

Referenced by initialise_basis().

Here is the call graph for this function:


basis_functions::reduced_hamiltonian_sto3g
component_functions::centered_gaussian_laplacian
basis_functions::wave_function_sto3g
component_functions::centered_gaussian

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::reduced_hamiltonian_sto3g

11.1.2.9 wave_function_2_electrons()

```
real(dp) function basis_functions::wave_function_2_electrons (
            real(dp), dimension(:), intent(in) position,
            real(dp), dimension(:), intent(in) dof_coefficients )
```

Wavefunction for 2 electrons.

Uses subprocedure correlation_function in component_functions.f90 Uses the function pointer wave_function_single

Parameters

in	*position*	Space coordinate of electrons
in	*dof_coefficients*	Values of the dofs

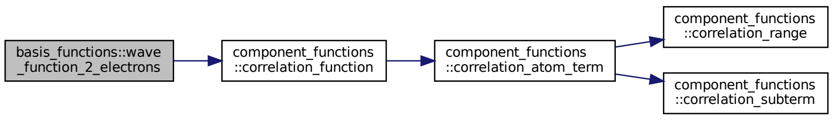
Definition at line 417 of file basis_functions.f90.

```
    implicit none
    real(dp) :: wave_function_2_electrons
    real(dp), dimension(:), intent(in) :: position
    real(dp), dimension(:), intent(in) :: dof_coefficients
    ! Initialise and bounds tests
    if (.not.(initialised)) then
      print *, "Error, basis not initialised"
      stop
    end if
    if (size(position).ne.n_space_dims) then
      print *, "Error, wrong space dimension, got", size(position), "wanted", n_space_dims
      stop
    end if
    if (size(dof_coefficients).ne.number_dofs) then
      print *, "Error, wrong number of dofs, got", size(dof_coefficients), "wanted", number_dofs
      stop
    end if

    ! Slater determinant (trivial for 2 electron spin up/down pair)
    ! position(1:3) is coords of first electron, (4:6) is the second. dof(:n_no_jastrow) are the
   relevant dofs
    wave_function_2_electrons =
   wave_function_single(position(1:3),dof_coefficients(:n_dofs_no_jastrow))&
    * wave_function_single(position(4:6),dof_coefficients(:n_dofs_no_jastrow))

    ! Multiply by correlation function. This is the Jastrow term. dof(n_dofs_no_Jastrow+1:) are relevant
   terms
    if (n_jastrow_dofs.ne.0) then
    wave_function_2_electrons = &
      correlation_function(atom_coords,position,mno_parameters,dof_coefficients(n_dofs_no_jastrow+1:),&
      b_length,d_length)*wave_function_2_electrons
    end if
```

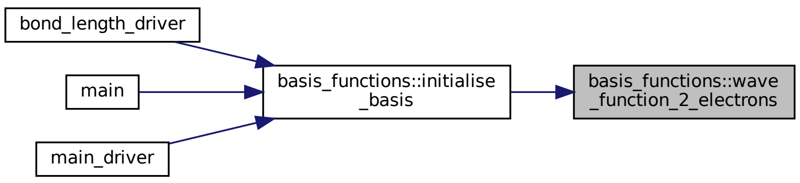
References atom_coords, b_length, component_functions::correlation_function(), d_length, initialised, mno_↩
parameters, n_dofs_no_jastrow, n_jastrow_dofs, n_space_dims, number_dofs, and wave_function_single.

Referenced by initialise_basis().

Here is the call graph for this function:


basis_functions::wave_function_2_electrons
component_functions::correlation_function
component_functions::correlation_atom_term
component_functions::correlation_range
component_functions::correlation_subterm

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::wave_function_2_electrons

11.1.2.10 wave_function_slater_1s()

```
real(dp) function basis_functions::wave_function_slater_1s (
            real(dp), dimension(:), intent(in) position,
            real(dp), dimension(:), intent(in) dof_coefficients )
```

Single Electron wavefunction: slater 1s basis.

Uses centered_slater_1s in component_functions.f90

Parameters

in	*position*	Space coordinate of electron
in	*dof_coefficients*	Values of the dofs

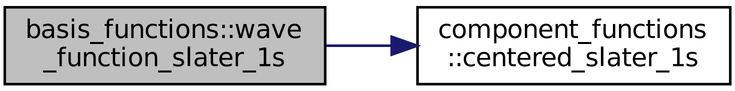
Definition at line 276 of file basis_functions.f90.

```
    implicit none
    real(dp) :: wave_function_slater_1s
    real(dp), dimension(:), intent(in) :: position
    real(dp), dimension(:), intent(in) :: dof_coefficients
    integer :: i, j ! loop variables
    ! Initialise and bounds tests
    if (.not.(initialised)) then
      print *, "Error, basis not initialised"
      stop
    end if
    if (size(position).ne.3) then
      print *, "Error, wrong space dimension, got", size(position), "wanted 3"
      stop
    end if
    if (size(dof_coefficients).ne.n_dofs_no_jastrow) then
      print *, "Error, wrong number of dofs, got", size(dof_coefficients), "wanted", n_dofs_no_jastrow
      stop
    end if

    ! Wavefunction is sum over linear terms with 2 dofs each - a linear coefficient and a lengthscale
    wave_function_slater_1s = 0.0_dp
    do j = 0, n_atoms-1 ! loop over atoms
      do i= 1, n_basis_functions_per_atom ! loop over linear terms in wavefunction
        wave_function_slater_1s = wave_function_slater_1s + &
          dof_coefficients( 2*i-1 + j*n_dofs_per_atom )& ! linear coefficient dof
            *centered_slater_1s(position-atom_coords(:,j+1), & ! position relative to atom
              dof_coefficients( 2*i + j*n_dofs_per_atom  ) ) ! lengthscale dof
      end do
    end do

```

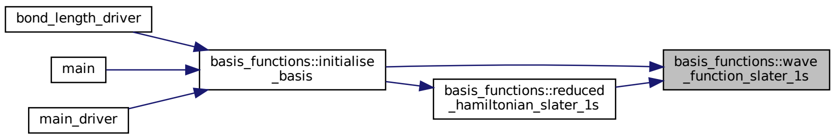
References atom_coords, component_functions::centered_slater_1s(), initialised, n_atoms, n_basis_functions_per_atom, n_dofs_no_jastrow, and n_dofs_per_atom.

Referenced by initialise_basis(), and reduced_hamiltonian_slater_1s().

Here is the call graph for this function:


basis_functions::wave_function_slater_1s
component_functions::centered_slater_1s

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::reduced_hamiltonian_slater_1s
basis_functions::wave_function_slater_1s

11.1.2.11 wave_function_sto3g()

```
real(dp) function basis_functions::wave_function_sto3g (
            real(dp), dimension(:), intent(in) position,
            real(dp), dimension(:), intent(in) dof_coefficients )
```

Single Electron wavefunction: gaussian sto3g basis.

Uses centered_gaussian in component_functions.f90

Parameters

in	*position*	Space coordinate of electron
in	*dof_coefficients*	Values of the dofs

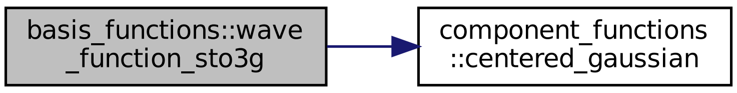
Definition at line 554 of file basis_functions.f90.

```
    implicit none
    real(dp) :: wave_function_sto3g
    real(dp), dimension(:), intent(in) :: position
    real(dp), dimension(:), intent(in) :: dof_coefficients

    integer :: i, j ! loop variables
    ! Initialise and bounds tests
    if (.not.(initialised)) then
      print *, "Error, basis not initialised"
      stop
    end if
    if (size(position).ne.3) then
      print *, "Error, wrong space dimension, got", size(position), "wanted 3"
      stop
    end if
    if (size(dof_coefficients).ne.n_dofs_no_jastrow) then
      print *, "Error, wrong number of dofs, got", size(dof_coefficients), "wanted", n_dofs_no_jastrow
      stop
    end if

    ! Wavefunction is sum over linear terms with 2 dofs each - a linear coefficient and a lengthscale
    wave_function_sto3g = 0.0_dp
    do j = 0, n_atoms-1 ! loop over atoms
      do i= 1, n_basis_functions_per_atom ! loop over linear terms in wavefunction
        wave_function_sto3g = wave_function_sto3g + &
          dof_coefficients( 2*i-1 + j*n_dofs_per_atom )& ! linear coefficient dof
            *centered_gaussian(position-atom_coords(:,j+1), & ! position relative to atom
              dof_coefficients( 2*i + j*n_dofs_per_atom  ) ) ! lengthscale dof
      end do
    end do

```

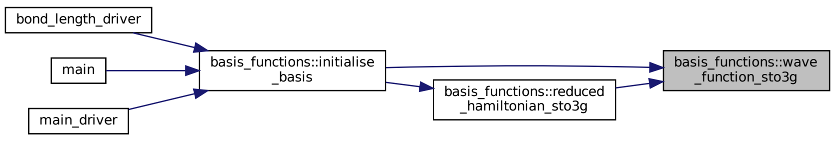
References atom_coords, component_functions::centered_gaussian(), initialised, n_atoms, n_basis_functions_↩
per_atom, n_dofs_no_jastrow, and n_dofs_per_atom.

Referenced by initialise_basis(), and reduced_hamiltonian_sto3g().

Here is the call graph for this function:



Here is the caller graph for this function:



11.1.3 Variable Documentation

11.1.3.1 atom_coords

```
real(dp), dimension(:,:), allocatable, protected basis_functions::atom_coords
```

Definition at line 42 of file basis_functions.f90.

```
42   real(dp), protected, allocatable, dimension(:,:) :: atom_coords ! Coordinates of the atoms. Shape
       (3,n_atoms)
```

Referenced by deinitialise_basis(), initialise_basis(), reduced_hamiltonian_2_electrons(), reduced_hamiltonian_↩
slater_1s(), reduced_hamiltonian_sto3g(), wave_function_2_electrons(), wave_function_slater_1s(), and wave_↩
function_sto3g().

11.1.3.2 b_length

```
real(dp), protected basis_functions::b_length
```

Definition at line 45 of file basis_functions.f90.

```
45   real(dp), protected :: b_length ! Inverse lengthscale of nuclear-electron interaction
```

Referenced by initialise_basis(), and wave_function_2_electrons().

11.1.3.3 basis_type

```
integer, protected basis_functions::basis_type
```

Definition at line 48 of file basis_functions.f90.

```
integer, protected :: basis_type ! integer code for type of basis. In shared_constants module
```

Referenced by initialise_basis().

11.1.3.4 d_length

```
real(dp), protected basis_functions::d_length
```

Definition at line 46 of file basis_functions.f90.

```
real(dp), protected :: d_length ! Inverse lengthscale of electron-electron interaction
```

Referenced by initialise_basis(), and wave_function_2_electrons().

11.1.3.5 dof_bounds

```
real(dp), dimension(:,:), allocatable, protected basis_functions::dof_bounds
```

Definition at line 52 of file basis_functions.f90.

```
real(dp), allocatable, dimension(:,:), protected :: dof_bounds ! Default bounds for the possible dof
  values
```

Referenced by bond_length_driver(), deinitialise_basis(), stoch_grad::grad_accent(), initialise_basis(), main(), main_driver(), and stoch_grad::stoch_grad_init().

11.1.3.6 fd_h

```
real(dp), protected basis_functions::fd_h
```

Definition at line 44 of file basis_functions.f90.

```
real(dp), protected :: fd_h ! Lengthscale of the finite difference code
```

Referenced by initialise_basis(), and reduced_hamiltonian_2_electrons().

11.1.3.7 initialised

```
logical, protected basis_functions::initialised = .false.
```

Definition at line 36 of file basis_functions.f90.

```
logical, protected :: initialised = .false. ! Flag set by initialisation routine, checked by other
  routines
```

Referenced by deinitialise_basis(), discrete_laplacian_reduced(), initialise_basis(), reduced_hamiltonian_2_↩
electrons(), reduced_hamiltonian_slater_1s(), reduced_hamiltonian_sto3g(), wave_function_2_electrons(), wave↩
_function_slater_1s(), and wave_function_sto3g().

11.1.3.8 mno_parameters

`integer, dimension(:,:), allocatable, protected basis_functions::mno_parameters`

Definition at line 57 of file basis_functions.f90.

```
integer, protected, allocatable, dimension(:,:) :: mno_parameters ! Parameters that specify the
   particular choice of Jastrow term
```

Referenced by deinitialise_basis(), mno_allocate(), and wave_function_2_electrons().

11.1.3.9 n_atoms

`integer, protected basis_functions::n_atoms`

Definition at line 41 of file basis_functions.f90.

```
integer, protected :: n_atoms ! Number of atoms
```

Referenced by initialise_basis(), reduced_hamiltonian_2_electrons(), reduced_hamiltonian_slater_1s(), reduced←_hamiltonian_sto3g(), wave_function_slater_1s(), and wave_function_sto3g().

11.1.3.10 n_basis_functions_per_atom

`integer, protected basis_functions::n_basis_functions_per_atom`

Definition at line 40 of file basis_functions.f90.

```
integer, protected :: n_basis_functions_per_atom ! Number of linear terms in the single-electron
   wavefunction per atom
```

Referenced by initialise_basis(), reduced_hamiltonian_slater_1s(), reduced_hamiltonian_sto3g(), wave_function←_slater_1s(), and wave_function_sto3g().

11.1.3.11 n_dofs_no_jastrow

`integer, protected basis_functions::n_dofs_no_jastrow`

Definition at line 56 of file basis_functions.f90.

```
integer, protected :: n_dofs_no_Jastrow ! Number of dofs without Jastrow term, for convenience
```

Referenced by initialise_basis(), wave_function_2_electrons(), wave_function_slater_1s(), and wave_function_←sto3g().

11.1.3.12 n_dofs_per_atom

```
integer, protected basis_functions::n_dofs_per_atom
```

Definition at line 55 of file basis_functions.f90.

```
integer, protected :: n_dofs_per_atom ! Number of dofs per atom, for convenience
```

Referenced by initialise_basis(), reduced_hamiltonian_slater_1s(), reduced_hamiltonian_sto3g(), wave_function←_slater_1s(), and wave_function_sto3g().

11.1.3.13 n_electrons

```
integer, protected basis_functions::n_electrons
```

Definition at line 39 of file basis_functions.f90.

```
integer, protected :: n_electrons ! Number of electrons
```

Referenced by initialise_basis().

11.1.3.14 n_jastrow_dofs

```
integer, protected basis_functions::n_jastrow_dofs
```

Definition at line 43 of file basis_functions.f90.

```
integer, protected :: n_Jastrow_dofs ! Number of dofs in the Jastrow (interaction) term
```

Referenced by initialise_basis(), mno_allocate(), and wave_function_2_electrons().

11.1.3.15 n_space_dims

```
integer, protected basis_functions::n_space_dims
```

Definition at line 51 of file basis_functions.f90.

```
integer, protected :: n_space_dims ! Number of space dimensions, 3*n_electrons, for convenience
```

Referenced by bond_length_driver(), discrete_laplacian_reduced(), initialise_basis(), main_driver(), reduced_←hamiltonian_2_electrons(), reduced_hamiltonian_slater_1s(), reduced_hamiltonian_sto3g(), wave_function_2_←electrons(), and electron_density_functions::wave_function_normalisation().

11.1.3.16 number_dofs

```
integer, protected basis_functions::number_dofs
```

Definition at line 50 of file basis_functions.f90.

```
50   integer, protected :: number_dofs ! Total number of degrees of freedom (dofs) in the wavefunction
```

Referenced by bond_length_driver(), discrete_laplacian_reduced(), initialise_basis(), main(), main_driver(), reduced_hamiltonian_2_electrons(), reduced_hamiltonian_slater_1s(), reduced_hamiltonian_sto3g(), and wave_↩ function_2_electrons().

11.1.3.17 proton_numbers

```
real(dp), dimension(:), allocatable, protected basis_functions::proton_numbers
```

Definition at line 47 of file basis_functions.f90.

```
47   real(dp),dimension(:),allocatable, protected :: proton_numbers ! Proton Numbers of atoms
```

Referenced by deinitialise_basis(), initialise_basis(), reduced_hamiltonian_2_electrons(), reduced_hamiltonian_↩ slater_1s(), and reduced_hamiltonian_sto3g().

11.1.3.18 reduced_hamiltonian

```
procedure(wave_function_interface), pointer basis_functions::reduced_hamiltonian
```

Definition at line 21 of file basis_functions.f90.

```
21   procedure(wave_function_interface), pointer :: reduced_hamiltonian
```

Referenced by bond_length_driver(), initialise_basis(), main(), and main_driver().

11.1.3.19 wave_function

```
procedure(wave_function_interface), pointer basis_functions::wave_function
```

Definition at line 20 of file basis_functions.f90.

```
20   procedure(wave_function_interface), pointer :: wave_function
```

Referenced by calculations::calc(), discrete_laplacian_reduced(), electron_density_functions::electron_density(), initialise_basis(), log_density(), log_rho_mod::log_rho(), and electron_density_functions::wave_function_↩ normalisation().

11.1.3.20 wave_function_single

```
procedure(wave_function_interface), pointer basis_functions::wave_function_single
```

Definition at line 60 of file basis_functions.f90.

```
60    procedure(wave_function_interface), pointer :: wave_function_single
```

Referenced by initialise_basis(), and wave_function_2_electrons().

11.2 biased_optim Module Reference

Biased optimization subroutines and functions.

Data Types

- interface bi_op_init

Functions/Subroutines

- logical function find_restart_file ()
- subroutine bi_op_init_constant_mean (param_init_data, energy_init_data, n_data_in, n_dof_in, ker_var, ker_lengthscale, constant_mean_prior, optim_rate_para, optim_no_samples, n_threads, n_loops_to_do)
- subroutine bi_op_init_arb_mean (param_init_data, energy_init_data, n_data_in, n_dof_in, ker_var, ker_↩ lengthscale, mean_prior_func, mean_prior_dx, optim_rate_para, optim_no_samples, n_threads, n_loops_↩ to_do)
- real(dp) function, dimension(threads, n_dof) bi_op_step (param_update_data, energy_update_data, n_↩ new_data, threads, seed, n_cycles)

Variables

- logical gp_uptodate = .False.

11.2.1 Detailed Description

Biased optimization subroutines and functions.

11.2.2 Function/Subroutine Documentation

11.2.2.1 bi_op_init_arb_mean()

```
subroutine biased_optim::bi_op_init_arb_mean (
             real(dp), dimension(n_data_in, n_dof_in), intent(in) param_init_data,
             real(dp), dimension(n_data_in), intent(in) energy_init_data,
             integer, intent(in) n_data_in,
             integer, intent(in) n_dof_in,
             real(dp), intent(in) ker_var,
             real(dp), intent(in) ker_lengthscale,
             procedure(mean_func_interface) mean_prior_func,
             procedure(mean_func_interface_dx) mean_prior_dx,
             real(dp), intent(in) optim_rate_para,
             integer, intent(in) optim_no_samples,
             integer, intent(in) n_threads,
             integer, intent(inout) n_loops_to_do )
```

Definition at line 134 of file Biased_Optim.f90.

```
      implicit none
      integer, intent(inout) :: n_loops_to_do
      integer, intent(in) :: n_data_in, n_dof_in, optim_no_samples,n_threads
      real(dp), dimension(n_data_in, n_dof_in), intent(in) :: param_init_data
      real(dp), dimension(n_data_in), intent(in) :: energy_init_data
      real(dp), intent(in) :: ker_var, ker_lengthscale,  optim_rate_para
      procedure(mean_func_interface) :: mean_prior_func
      procedure(mean_func_interface_dx) :: mean_prior_dx

      n_dof = n_dof_in
      gamma = optim_rate_para
      no_samples=optim_no_samples

      call gp_init(mean_prior_func, mean_prior_dx, ker_var, ker_lengthscale, param_init_data,
energy_init_data,&
       n_data_in, n_dof_in, n_threads)
      gp_uptodate = .false.

      current_best_e = minval(energy_init_data)

```

References gp_uptodate.

11.2.2.2 bi_op_init_constant_mean()

```
subroutine biased_optim::bi_op_init_constant_mean (
             real(dp), dimension(n_data_in, n_dof_in), intent(in) param_init_data,
             real(dp), dimension(n_data_in), intent(in) energy_init_data,
             integer, intent(in) n_data_in,
             integer, intent(in) n_dof_in,
             real(dp), intent(in) ker_var,
             real(dp), intent(in) ker_lengthscale,
             real(dp), intent(in) constant_mean_prior,
             real(dp), intent(in) optim_rate_para,
             integer, intent(in) optim_no_samples,
             integer, intent(in) n_threads,
             integer, intent(inout) n_loops_to_do )
```

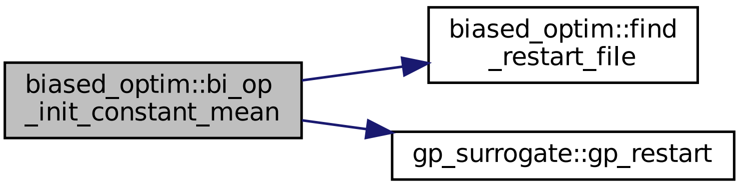
Definition at line 84 of file Biased_Optim.f90.

```
      implicit none
      integer, intent(inout) :: n_loops_to_do
      integer, intent(in) :: n_data_in, n_dof_in, optim_no_samples,n_threads
      real(dp), dimension(n_data_in, n_dof_in), intent(in) :: param_init_data
      real(dp), dimension(n_data_in), intent(in) :: energy_init_data
      real(dp), intent(in) :: ker_var, ker_lengthscale, constant_mean_prior, optim_rate_para
      integer gp_n_data, gp_n_dof, n_cycles
```

```
        real(dp) :: kernel_var, kernal_inv_length
        real(dp), dimension(:,:), allocatable :: param_data
        real(dp), dimension(:), allocatable :: E_data
        real(dp), dimension(:,:), allocatable :: param_pres, param_cov
        real(dp), dimension(:), allocatable :: data_mean

        if (find_restart_file()) then
            call read_restart_file_sizes(gp_n_data, gp_n_dof)
            allocate(param_data(gp_n_data, gp_n_dof))
             allocate(e_data(gp_n_data))
             allocate(param_pres(gp_n_data, gp_n_data))
             allocate(param_cov(gp_n_data, gp_n_data))
             allocate(data_mean(gp_n_data))
             call read_restart_file_data(n_cycles, no_samples, current_best_e,&
             constant_mean_value, kernel_var, gamma, kernal_inv_length,&
             gp_n_data, e_data, gp_n_data, data_mean, gp_n_data, gp_n_data, param_pres, &
             gp_n_data, gp_n_data, param_pres, gp_n_data, gp_n_dof, param_data)
             n_dof = gp_n_dof

             call gp_restart(gp_n_data, gp_n_dof,kernel_var, kernal_inv_length,&
             param_data, e_data,param_pres, param_cov, data_mean, constant_mean,&
              zero_func, n_threads)
             gp_uptodate = .false.
             current_best_e = minval(e_data)
             n_loops_to_do=n_loops_to_do-(n_cycles-1)
         else
             constant_mean_value = constant_mean_prior
             gamma = optim_rate_para
             no_samples=optim_no_samples
             n_dof = n_dof_in

             call gp_init(constant_mean, zero_func, ker_var, ker_lengthscale, param_init_data,
    energy_init_data, n_data_in,&
              n_dof_in,n_threads)
             gp_uptodate = .false.

             current_best_e = minval(energy_init_data)
         end if

```

References find_restart_file(), gp_surrogate::gp_restart(), and gp_uptodate.

Here is the call graph for this function:


biased_optim::bi_op_init_constant_mean
biased_optim::find_restart_file
gp_surrogate::gp_restart

11.2.2.3 bi_op_step()

```
real(dp) function, dimension(threads,n_dof) biased_optim::bi_op_step (
            real(dp), dimension(n_new_data, n_dof), intent(in) param_update_data,
            real(dp), dimension(n_new_data), intent(in) energy_update_data,
            integer, intent(in) n_new_data,
            integer, intent(in) threads,
            integer, dimension(:), intent(in) seed,
            integer, intent(in) n_cycles )
```

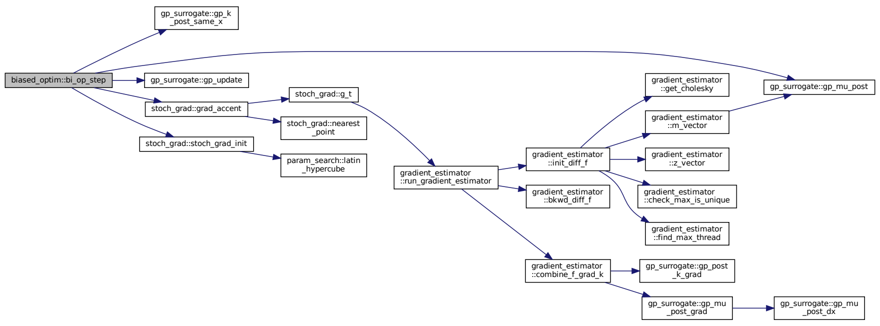
Definition at line 157 of file Biased_Optim.f90.

```
157         implicit none
158         integer, intent(in) :: n_new_data, n_cycles
159         integer, intent(in) :: threads !number of threads/restarts goes here
160         integer, dimension(:), intent(IN):: seed
161         real(dp), dimension(n_new_data, n_dof), intent(in) :: param_update_data
162         real(dp), dimension(n_new_data), intent(in) :: energy_update_data
163         real(dp), dimension(threads,threads, n_dof) ::  per_thread_best_loc
164         real(dp), dimension(0:threads) ::  m,z
165         real(dp), dimension(0:threads,0:threads) ::  c
166         real(dp), dimension(threads) ::  per_thread_best_qei
167         real(dp), dimension(threads,n_dof) :: best_locs
168         real(dp) :: best_E_in_batch
169         integer :: i, j, best_restart, n_seed, gp_n_data, gp_n_dof
170         real(dp) :: kernel_var, kernal_inv_length
171         real(dp), dimension(:,:), allocatable :: param_data
172         real(dp), dimension(:), allocatable :: E_data
173         real(dp), dimension(:,:), allocatable :: param_pres, param_cov
174         real(dp), dimension(:), allocatable :: data_mean
175 
176         if (gp_uptodate) then
177             call gp_update(param_update_data,  energy_update_data, n_new_data, 'F')
178         end if
179 
180         !updating best point
181         best_e_in_batch = minval(energy_update_data)
182         if (best_e_in_batch < current_best_e) current_best_e=best_e_in_batch
183         print*, current_best_e
184 
185         call stoch_grad_init(threads,current_best_e,seed)
186 
187         ! !doing the restart file thing
188         ! call gp_return_size_data(gp_n_data, gp_n_dof)
189         ! allocate(param_data(gp_n_data, gp_n_dof))
190         ! allocate(E_data(gp_n_data))
191         ! allocate(param_pres(gp_n_data, gp_n_data))
192         ! allocate(param_cov(gp_n_data, gp_n_data))
193         ! allocate(data_mean(gp_n_data))
194         ! call gp_return_state_data(kernel_var, kernal_inv_length,  param_data, E_data,&
195         ! param_pres, param_cov, data_mean)
196         ! !assuming calling order of (scalars(rank 0),..., vector_dim, vector(rank 1),...,
       matrix_dim_1,matrix_dim2, matrix(rank 2))
197         ! call write_restart_file(gp_n_data, gp_n_dof,n_cycles,no_samples, current_best_E,&
198         ! constant_mean_value, gamma, kernel_var, kernal_inv_length, &
199         ! gp_n_data, E_data, gp_n_data, data_mean, gp_n_data, gp_n_data, param_pres, &
200         ! gp_n_data, gp_n_data, param_pres, gp_n_data, gp_n_dof, param_data)
201 
202 
203         !doing grad_accent, can be parrallel
204         !$OMP parallel do default(shared) private(i,m,c,z)
205         do i=1,threads
206             !the "no change" state
207             z=0.0_dp
208             m=0.0_dp
209             call grad_accent(n_points(i,:,:),threads,no_samples,gamma,per_thread_best_loc(i,:,:))
210             m(1:threads) = current_best_e-gp_mu_post(per_thread_best_loc(i,:,:), threads)
211             call get_cholesky(c, gp_k_post_same_x(per_thread_best_loc(i,:,:), threads))
212             c= -c
213             !c = -get_cholesky(gp_k_post_same_x(per_thread_best_loc(i,:,:), threads))
214             do j=1,no_samples
215                 z(1:threads) = gaussian(threads)
216                 per_thread_best_qei(i) =  maxval(m + matmul(c, z))
217             end do
218         end do
219 
220 
221         best_restart = maxloc(per_thread_best_qei,dim=1)
222 
223         best_locs = per_thread_best_loc(best_restart,:,:)
224 
225         gp_uptodate = .true.
226 
```

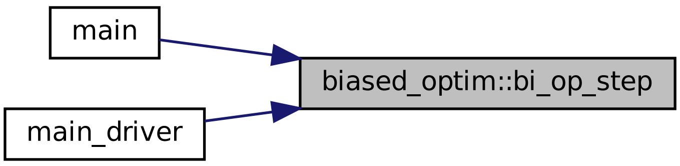
References gp_surrogate::gp_k_post_same_x(), gp_surrogate::gp_mu_post(), gp_surrogate::gp_update(), gp_↩ uptodate, stoch_grad::grad_accent(), stoch_grad::n_points, and stoch_grad::stoch_grad_init().

Referenced by main(), and main_driver().

Here is the call graph for this function:


biased_optim::bi_op_step
gp_surrogate::gp_k_post_same_x
gp_surrogate::gp_update
stoch_grad::grad_accent
stoch_grad::stoch_grad_init
stoch_grad::g_t
stoch_grad::nearest_point
param_search::latin_hypercube
gradient_estimator::run_gradient_estimator
gradient_estimator::init_diff_f
gradient_estimator::bkwd_diff_f
gradient_estimator::combine_f_grad_k
gradient_estimator::get_cholesky
gradient_estimator::m_vector
gradient_estimator::z_vector
gradient_estimator::check_max_is_unique
gradient_estimator::find_max_thread
gp_surrogate::gp_post_k_grad
gp_surrogate::gp_mu_post_grad
gp_surrogate::gp_mu_post_dx
gp_surrogate::gp_mu_post

Here is the caller graph for this function:



11.2.2.4 find_restart_file()

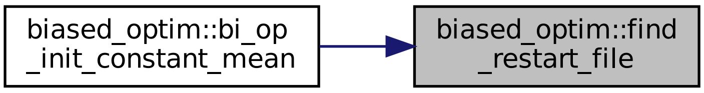
```
logical function biased_optim::find_restart_file
```

Definition at line 74 of file Biased_Optim.f90.

```
74         implicit none
75         logical :: retval
76
77         retval=.false.
78
```

Referenced by bi_op_init_constant_mean().

Here is the caller graph for this function:


biased_optim::bi_op_init_constant_mean
biased_optim::find_restart_file

11.2.3 Variable Documentation

11.2.3.1 gp_uptodate

```
logical biased_optim::gp_uptodate = .False.
```

Definition at line 15 of file Biased_Optim.f90.

```
15      logical :: GP_uptodate = .false.
```

Referenced by bi_op_init_arb_mean(), bi_op_init_constant_mean(), and bi_op_step().

11.3 calculations Module Reference

Function completing calculations of either the wavefunction or electron density (depending on the system) at a set of coordinates [x,y,0] on a grid of equally spaced points on a xy grid.

Functions/Subroutines

- real(dp) function, dimension(:), allocatable calc (dof, points, box, n_ele, n_MCMC_steps)

  calc function returns the wavefunction or electron density evaluated on a 2D xy plane (taking a slice) depending on the number of electrons in the system.

11.3.1 Detailed Description

Function completing calculations of either the wavefunction or electron density (depending on the system) at a set of coordinates [x,y,0] on a grid of equally spaced points on a xy grid.

11.3.2 Function/Subroutine Documentation

11.3.2.1 calc()

```
real(dp) function, dimension(:), allocatable calculations::calc (
            real(dp), dimension(:), intent(in) dof,
            integer, intent(in) points,
            real(dp), intent(in) box,
            integer, intent(in) n_ele,
            integer, intent(in) n_MCMC_steps )
```

calc function returns the wavefunction or electron density evaluated on a 2D xy plane (taking a slice) depending on the number of electrons in the system.

Sets the z coordinate to 0 as only evaluating on a xy plane.

Parameters

in	*points*	points is the user defined number of evaluation points for the plotting grid.
	n_ele	is the number of electrons in the system
	n_MCMC_steps	is the number of steps needed for the MCMC used for evaluating the electron density for the 2 electron case.
in	*n_ele*	points is the user defined number of evaluation points for the plotting grid.
	n_ele	is the number of electrons in the system
	n_MCMC_steps	is the number of steps needed for the MCMC used for evaluating the electron density for the 2 electron case.
in	*n_mcmc_steps*	points is the user defined number of evaluation points for the plotting grid.
	n_ele	is the number of electrons in the system
	n_MCMC_steps	is the number of steps needed for the MCMC used for evaluating the electron density for the 2 electron case.
in	*box*	box is the user defined size of the domain of the system wanting to be looked at.
in	*dof*	dof is the optimal degrees of freedom

Returns

Parameters

calc	Ouput array which contains the results of the function

This allocates memory to the array based on the number of grid points defined by the user.

Uses wave_function_normalisation which returns the wave_norm_squared

This is for the 1 electron case, will just calculate and returns the wavefunction. Uses the wave_function function.

Track puts the result in the right place in the resulting output calc array.

Calculating the positions for input and gives coordinates between -box/2 to box/2

Returns the result of the wavefunction evaluated at one coordinate point using wave_function function.

Divides the result by the square root of the results from the wave_norm_squared result defined earlier.

For the 2 electron case, will calculate and return the electron density. Uses the electron_density function

Calculating the positions for input

Returns the result of the electron density evaluated at one coordinate point using electron_density function.

Divides the result by the results from the wave_norm_squared result defined earlier.

Returns a error print statement if the number of electrons is not defined.

Definition at line 18 of file calc.f90.

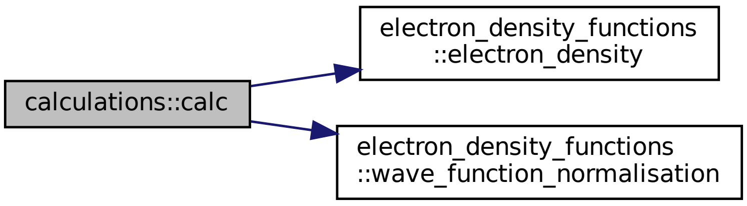
```
18
19     IMPLICIT NONE
20     INTEGER, PARAMETER :: dp=kind(1.0d0)
24     INTEGER, INTENT(IN) :: points, n_ele, n_MCMC_steps
26     real(dp),intent(in) :: box
```

```
    REAL(dp), DIMENSION(:), INTENT(IN) :: dof
    INTEGER :: array_len, track, i,j
    REAL(dp) :: a,b
    real(dp),dimension(3,2) :: box_in
    real(dp) :: wave_norm_squared
    REAL(dp), DIMENSION(:), ALLOCATABLE :: calc

    do i=1,3
      box_in(i,:) = [-box,box]
    end do

    array_len = (abs(points) + abs(points) + 1)**2
    ALLOCATE(calc(array_len))

    wave_norm_squared = wave_function_normalisation(box_in,dof,n_mcmc_steps)
    track = 0

    IF (n_ele .eq. 1) THEN

      DO i = -points, points
        DO j = -points, points
          track = track + 1
          a =  (real(i) / (abs(points) + abs(points)))*box
          b =  (real(j) / (abs(points) + abs(points)))*box
          calc(track) = wave_function([a,b,0.0d0], dof)
        END DO
      END DO
      calc = calc/sqrt(wave_norm_squared)

    ELSE IF (n_ele .eq. 2) THEN

      DO i = -points, points
        DO j = -points, points
          track = track + 1
          a =  (real(i) / (abs(points) + abs(points)))*box
          b =  (real(j) / (abs(points) + abs(points)))*box
          calc(track) = electron_density([a,b,0.0d0],box_in, dof, n_mcmc_steps )
        END DO
      END DO
      calc = calc/wave_norm_squared

    ELSE
      print*, 'Incorrect number of electrons in output calculation'
    END IF

```

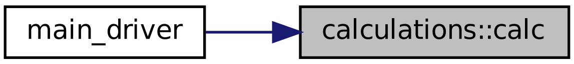
References electron_density_functions::electron_density(), basis_functions::wave_function, and electron_density_functions::wave_function_normalisation().

Referenced by main_driver().

Here is the call graph for this function:


calculations::calc
electron_density_functions ::electron_density
electron_density_functions ::wave_function_normalisation

Here is the caller graph for this function:



11.4 component_functions Module Reference

Basic subfunctions This module contains basic functions used in basis_functions.f90.

Functions/Subroutines

- real(dp) function centered_gaussian (position, alpha)
  Gaussian distribution centred at the origin.
- real(dp) function centered_gaussian_laplacian (position, alpha)
  Analytic Laplacian of Gaussian distribution centred at the origin.
- real(dp) function centered_slater_1s (position, zeta)
  Slater-1s distribution centred at the origin.
- real(dp) function centered_slater_1s_laplacian (position, zeta)
  Analytic Laplacian of Slater-1s distribution centred at the origin.
- real(dp) function correlation_range (r, d)
  Functions that construct the Jastrow interaction function Notation here follows Schmidt and Moskowitz 1990, refered to as SM90.
- real(dp) function correlation_subterm (r_12, r_l1, r_l2, m, n, o)
  Jastrow subfuction: basic subterm Subterm of the correlation function, one for each Jastrow dof per atom This is the term in the k sum in Schmidt and Moskowitz 1990.
- real(dp) function correlation_atom_term (atom_coord, electron_coords, mno_parameters, c, b, d)
  Jastrow subfuction: atom subterm Correlation term for each atom.
- real(dp) function correlation_function (atom_coords, electron_coords, mno_parameters, c, b, d)
  Jastrow correlation fuction Function F from Schmidt and Moskowitz 1990.

11.4.1 Detailed Description

Basic subfunctions This module contains basic functions used in basis_functions.f90.

11.4.2 Function/Subroutine Documentation

11.4.2.1 centered_gaussian()

```
real(dp) function component_functions::centered_gaussian (
            real(dp), dimension(3), intent(in) position,
            real(dp), intent(in) alpha )
```

Gaussian distribution centred at the origin.

Parameters

in	*position*	Space Coordinate
in	*alpha*	Length scale

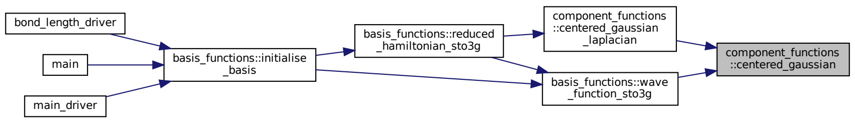
Definition at line 11 of file component_functions.f90.

```
implicit none
real(dp) :: centered_gaussian
real(dp), dimension(3), intent(in) :: position
real(dp), intent(in) :: alpha
centered_gaussian = (2.0_dp * alpha /pi)**(0.75_dp)*exp(-alpha * dot_product(position,position))
```

References shared_constants::pi.

Referenced by centered_gaussian_laplacian(), and basis_functions::wave_function_sto3g().

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::reduced_hamiltonian_sto3g
component_functions::centered_gaussian_laplacian
basis_functions::wave_function_sto3g
component_functions::centered_gaussian

11.4.2.2 centered_gaussian_laplacian()

```
real(dp) function component_functions::centered_gaussian_laplacian (
            real(dp), dimension(3), intent(in) position,
            real(dp), intent(in) alpha )
```

Analytic Laplacian of Gaussian distribution centred at the origin.

Parameters

in	*position*	Space Coordinate
in	*alpha*	Length scale

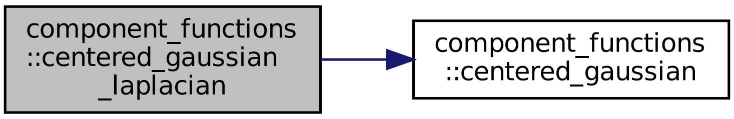
Definition at line 22 of file component_functions.f90.

```
implicit none
real(dp) :: centered_gaussian_laplacian
real(dp), dimension(3), intent(in) :: position
real(dp), intent(in) :: alpha

centered_gaussian_laplacian = &
(-6.0_dp*alpha + 4.0_dp*alpha**2*dot_product(position,position))*centered_gaussian(position,alpha)
```

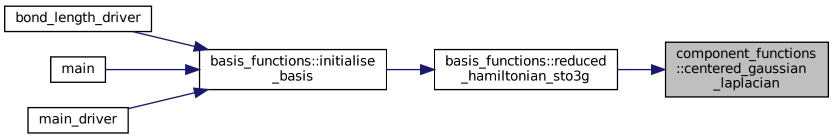
References centered_gaussian().

Referenced by basis_functions::reduced_hamiltonian_sto3g().

Here is the call graph for this function:


component_functions ::centered_gaussian _laplacian
component_functions ::centered_gaussian

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise _basis
basis_functions::reduced _hamiltonian_sto3g
component_functions ::centered_gaussian _laplacian

11.4.2.3 centered_slater_1s()

```
real(dp) function component_functions::centered_slater_1s (
            real(dp), dimension(3), intent(in) position,
            real(dp), intent(in) zeta )
```

Slater-1s distribution centred at the origin.

Parameters

in	*position*	Space Coordinate
in	*zeta*	Length scale

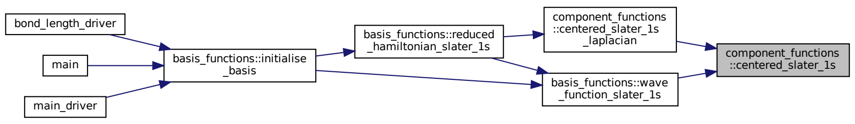
Definition at line 35 of file component_functions.f90.

```
      implicit none
      real(dp) :: centered_slater_1s
      real(dp), dimension(3), intent(in) :: position
      real(dp),intent(in) :: zeta

      centered_slater_1s = ( zeta**3 /pi)**(0.5_dp)*exp(-zeta * norm2(position))
```

References shared_constants::pi.

Referenced by centered_slater_1s_laplacian(), and basis_functions::wave_function_slater_1s().

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::reduced_hamiltonian_slater_1s
component_functions::centered_slater_1s_laplacian
basis_functions::wave_function_slater_1s
component_functions::centered_slater_1s

11.4.2.4 centered_slater_1s_laplacian()

```
real(dp) function component_functions::centered_slater_1s_laplacian (
            real(dp), dimension(3), intent(in) position,
            real(dp), intent(in) zeta )
```

Analytic Laplacian of Slater-1s distribution centred at the origin.

Parameters

in	*position*	Space Coordinate
in	*zeta*	Length scale

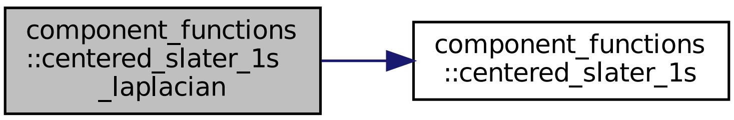
Definition at line 47 of file component_functions.f90.

```
47
48     implicit none
49     real(dp) :: centered_slater_1s_laplacian
51     real(dp), dimension(3), intent(in) :: position
53     real(dp), intent(in) :: zeta
54
55     centered_slater_1s_laplacian = &
56     (-2.0_dp*zeta/norm2(position) + zeta**2)*centered_slater_1s(position,zeta)
```

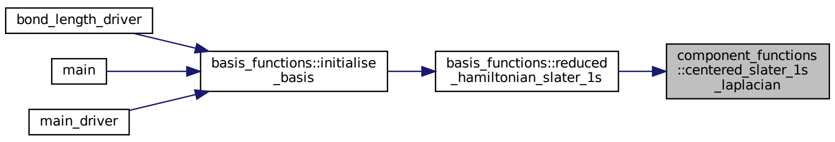
References centered_slater_1s().

Referenced by basis_functions::reduced_hamiltonian_slater_1s().

Here is the call graph for this function:


component_functions::centered_slater_1s_laplacian
component_functions::centered_slater_1s

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise _basis
basis_functions::reduced _hamiltonian_slater_1s
component_functions ::centered_slater_1s _laplacian

11.4.2.5 correlation_atom_term()

```
real(dp) function component_functions::correlation_atom_term (
            real(dp), dimension(3), intent(in) atom_coord,
            real(dp), dimension(6), intent(in) electron_coords,
            integer, dimension(:,:), intent(in) mno_parameters,
            real(dp), dimension(:), intent(in) c,
            real(dp), intent(in) b,
            real(dp), intent(in) d )
```

Jastrow subfuction: atom subterm Correlation term for each atom.

U_I12 from Schmidt and Moskowitz 1990

Parameters

in	*atom_coord*	Space coordinates of atom
in	*electron_coords*	Space positions of electrons
in	*mno_parameters*	Jastrow interger parameters
in	*c*	Jastrow dofs
in	*b*	Inverse lengthscale of nuclear-electron interaction
in	*d*	Inverse lengthscale of electron-electron interaction

Definition at line 105 of file component_functions.f90.

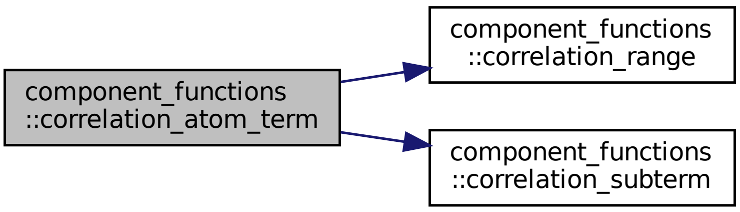
```

    implicit none
    real(dp) :: correlation_atom_term
    real(dp), dimension(3), intent(in) :: atom_coord
    real(dp), dimension(6), intent(in) :: electron_coords
    integer, dimension(:,:), intent(in) :: mno_parameters
    real(dp), dimension(:), intent(in) :: c
    real(dp), intent(in) :: b
    real(dp), intent(in) :: d

    ! Internal variables
    real(dp) :: r_12 ! Distance between electrons
    real(dp) :: r_I1 ! Distance from atom to electron 1
    real(dp) :: r_I2 ! Distance from atom to electron 2
    integer :: k ! Loop variable

    ! Compute distances
    r_12 = correlation_range(norm2(electron_coords(1:3)-electron_coords(4:6)),d)
    r_i1 = correlation_range(norm2(atom_coord-electron_coords(1:3)),b)
    r_i2 = correlation_range(norm2(atom_coord-electron_coords(4:6)),b)

    ! k sum from Schmidt and Moskowitz 1990 to compute U_I12
    correlation_atom_term = 0.0_dp
    do k = 1, size(mno_parameters,2)
```

```
      correlation_atom_term = correlation_atom_term + &
      c(k)*correlation_subterm(r_12,r_i1,r_i2,mno_parameters(1,k),mno_parameters(2,k)&
        ,mno_parameters(3,k))
    end do

```

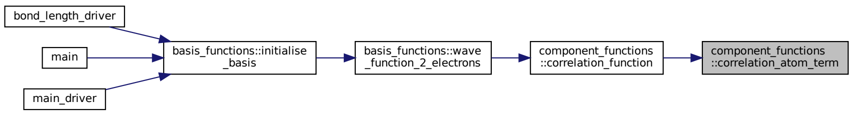
References correlation_range(), and correlation_subterm().

Referenced by correlation_function().

Here is the call graph for this function:


component_functions ::correlation_atom_term
component_functions ::correlation_range
component_functions ::correlation_subterm

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise _basis
basis_functions::wave _function_2_electrons
component_functions ::correlation_function
component_functions ::correlation_atom_term

11.4.2.6 correlation_function()

```
real(dp) function component_functions::correlation_function (
            real(dp), dimension(:,:), intent(in) atom_coords,
            real(dp), dimension(6), intent(in) electron_coords,
            integer, dimension(:,:), intent(in) mno_parameters,
            real(dp), dimension(:), intent(in) c,
            real(dp), intent(in) b,
            real(dp), intent(in) d )
```

Jastrow correlation fuction Function F from Schmidt and Moskowitz 1990.

Parameters

in	*atom_coords*	Coordinates of atoms
in	*electron_coords*	Coordinates of electrons
in	*mno_parameters*	Jastrow interger parameters
in	*c*	Jastrow dofs
in	*b*	Inverse lengthscale of nuclear-electron interaction
in	*d*	Inverse lengthscale of electron-electron interaction

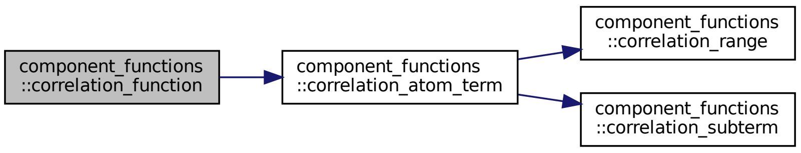
Definition at line 145 of file component_functions.f90.

```
    real(dp) :: correlation_function
    real(dp), dimension(:,:), intent(in) :: atom_coords
    real(dp), dimension(6), intent(in) :: electron_coords
    integer, dimension(:,:), intent(in) :: mno_parameters
    real(dp),dimension(:), intent(in) :: c
    real(dp), intent(in) :: b
    real(dp), intent(in) :: d

    integer :: i ! Loop variable

    correlation_function = 0.0_dp
    do i = 1, size(atom_coords,2) ! sum over atoms
      correlation_function = correlation_function +&
       correlation_atom_term(atom_coords(:,i),electron_coords,mno_parameters,c,b,d)
    end do
    ! Compute exponential
    correlation_function = exp(correlation_function)
```

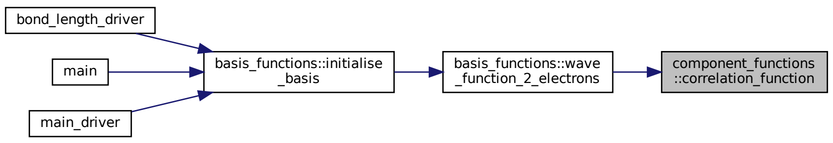
References correlation_atom_term().

Referenced by basis_functions::wave_function_2_electrons().

Here is the call graph for this function:


component_functions ::correlation_function
component_functions ::correlation_atom_term
component_functions ::correlation_range
component_functions ::correlation_subterm

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise _basis
basis_functions::wave _function_2_electrons
component_functions ::correlation_function

11.4.2.7 correlation_range()

```
real(dp) function component_functions::correlation_range (
              real(dp), intent(in) r,
              real(dp), intent(in) d )
```

Functions that construct the Jastrow interaction function Notation here follows Schmidt and Moskowitz 1990, refered to as SM90.

Jastrow subfuction: correlation range $\bar{r}$ function in Schmidt and Moskowitz 1990

Parameters

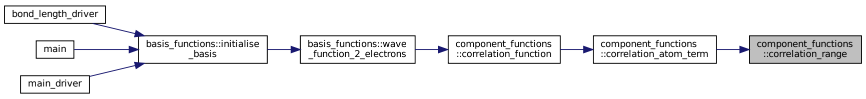
in	*r*	distance
in	*d*	inverse length scale

Definition at line 65 of file component_functions.f90.

```
65      implicit none
66      real(dp) :: correlation_range
68      real(dp), intent(in) :: r
70      real(dp), intent(in) :: d
71
72      correlation_range = d * r/(1+d*r)
```

Referenced by correlation_atom_term().

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::wave_function_2_electrons
component_functions::correlation_function
component_functions::correlation_atom_term
component_functions::correlation_range

11.4.2.8 correlation_subterm()

```
real(dp) function component_functions::correlation_subterm (
            real(dp), intent(in) r_12,
            real(dp), intent(in) r_I1,
            real(dp), intent(in) r_I2,
            integer, intent(in) m,
            integer, intent(in) n,
            integer, intent(in) o )
```

Jastrow subfuction: basic subterm Subterm of the correlation function, one for each Jastrow dof per atom This is the term in the k sum in Schmidt and Moskowitz 1990.

Parameters

in	*r_12*	Distance between electrons
in	*r_i1*	Distance from atom to electron 1
in	*r_i2*	Distance from atom to electron 2
in	*m*	Triple of integer parameters. This is a row of mno_parameters
in	*n*	Triple of integer parameters. This is a row of mno_parameters
in	*o*	Triple of integer parameters. This is a row of mno_parameters

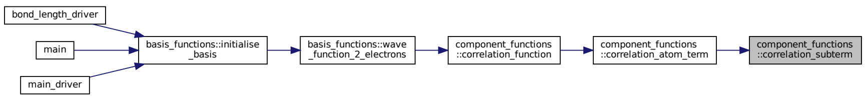
Definition at line 79 of file component_functions.f90.

```
79      implicit none
80      real(dp) :: correlation_subterm
82      real(dp), intent(in) :: r_12
84      real(dp), intent(in) :: r_I1
86      real(dp), intent(in) :: r_I2
88      integer,intent(in) :: m, n, o
89
```

```
    real(dp) :: delta ! coefficient to match Schmidt and Moskowitz 1990

    ! Value of delta, following Schmidt and Moskowitz 1990
    if (m==n) then
      delta = 0.5_dp
    else
      delta = 1.0_dp
    end if

    correlation_subterm = delta * (r_i1**m*r_i2**n+r_i1**n*r_i2**m)*r_12**o
```

Referenced by correlation_atom_term().

Here is the caller graph for this function:


bond_length_driver
main
main_driver
basis_functions::initialise_basis
basis_functions::wave_function_2_electrons
component_functions::correlation_function
component_functions::correlation_atom_term
component_functions::correlation_subterm

11.5 electron_density_functions Module Reference

Electron density function.

Functions/Subroutines

- real(dp) function electron_density (fixed_position, integral_bounds, dof_coefficients, n_MC_points, seed_in)
  Computes electron density for 2 electron simulations Uses basic Monte Carlo Integration.
- real(dp) function wave_function_normalisation (integral_bounds, dof_coefficients, n_MC_points, seed_in)
  Computes the integral of the wavefunction squared To normalise the wavefunction for outputing Uses basic Monte Carlo Integration.

11.5.1 Detailed Description

Electron density function.

11.5.2 Function/Subroutine Documentation

11.5.2.1 electron_density()

```
real(dp) function electron_density_functions::electron_density (
              real(dp), dimension(3), intent(in) fixed_position,
              real(dp), dimension(3,2), intent(in) integral_bounds,
              real(dp), dimension(:), intent(in) dof_coefficients,
              integer, intent(in) n_MC_points,
              integer, intent(in), optional seed_in )
```

Computes electron density for 2 electron simulations Uses basic Monte Carlo Integration.

Parameters

in	*fixed_position*	fixed_position Space position of density
in	*integral_bounds*	integral_bounds bounds of integration box
in	*dof_coefficients*	dof_coefficients Current dof parameters
in	*n_mc_points*	n_MC_points Number of Monte Carlo Points
in	*seed_in*	seed_in input seed

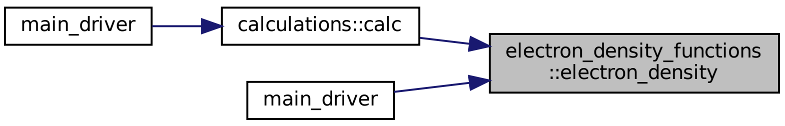
Definition at line 11 of file electron_density.f90.

```fortran
real(dp) :: electron_density
real(dp), dimension(3), intent(in) :: fixed_position
real(dp), dimension(3,2), intent(in) :: integral_bounds
real(dp), dimension(:), intent(in) :: dof_coefficients
integer, intent(in) :: n_MC_points
integer, intent(in),optional :: seed_in
integer, allocatable :: seed(:)
integer :: i ! loop variables
integer :: n
real(dp) :: x,y,z,Lx,Ly,Lz
real(dp), dimension(6) :: position_total
if (present(seed_in)) then
  call random_seed(size=n)
  allocate(seed(n))
  seed = seed_in
  call random_seed(put=seed)
end if

do i=1,3
if ((integral_bounds(i,2)-integral_bounds(i,1))<= 0.1) then
  print*, "Error, box too small or bounds incorrect"
end if
end do
position_total(1:3) = 0.0_dp
position_total(4:6) = fixed_position
lx = integral_bounds(1,2)-integral_bounds(1,1)
ly = integral_bounds(2,2)-integral_bounds(2,1)
lz = integral_bounds(3,2)-integral_bounds(3,1)
!$OMP parallel do default(shared) private(i,x,y,z) firstprivate(position_total)
reduction(+:electron_density)
do i=1,n_mc_points
  call random_number(x)
  call random_number(y)
  call random_number(z)
  x = lx*x +integral_bounds(1,1)
  y = ly*y +integral_bounds(2,1)
  z = lz*z +integral_bounds(3,1)
  position_total(4:6) = [x,y,z]
  electron_density = electron_density+wave_function(position_total,dof_coefficients)**2
end do
electron_density = 2*lx*ly*lz*electron_density/n_mc_points
if (allocated(seed)) deallocate(seed)
```

References basis_functions::wave_function.

Referenced by calculations::calc(), and main_driver().

Here is the caller graph for this function:



11.5.2.2 wave_function_normalisation()

```
real(dp) function electron_density_functions::wave_function_normalisation (
            real(dp), dimension(3,2), intent(in) integral_bounds,
            real(dp), dimension(:), intent(in) dof_coefficients,
            integer, intent(in) n_MC_points,
            integer, intent(in), optional seed_in )
```

Computes the integral of the wavefunction squared To normalise the wavefunction for outputing Uses basic Monte Carlo Integration.

Parameters

in	*integral_bounds*	bounds of integration box
in	*dof_coefficients*	Current dof parameters
in	*n_mc_points*	Number of Monte Carlo Points
in	*seed_in*	input seed

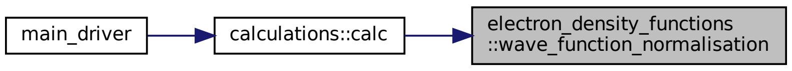
Definition at line 63 of file electron_density.f90.

```
real(dp) :: wave_function_normalisation
real(dp), dimension(3,2), intent(in) :: integral_bounds
real(dp), dimension(:), intent(in) :: dof_coefficients
integer, intent(in) :: n_MC_points
integer, intent(in),optional :: seed_in
integer, allocatable :: seed(:)
integer :: i ! loop variables
integer :: n
real(dp) :: x,y,z,Lx,Ly,Lz
real(dp), dimension(:),allocatable :: position_total
if (present(seed_in)) then
  call random_seed(size=n)
  allocate(seed(n))
  seed = seed_in
  call random_seed(put=seed)
end if

do i=1,3
if ((integral_bounds(i,2)-integral_bounds(i,1))<= 0.1) then
  print*, "Error, box too small or bounds incorrect"
end if
end do
allocate(position_total(n_space_dims))
lx = integral_bounds(1,2)-integral_bounds(1,1)
ly = integral_bounds(2,2)-integral_bounds(2,1)
lz = integral_bounds(3,2)-integral_bounds(3,1)
!$OMP parallel do default(shared) private(i,x,y,z) firstprivate(position_total)
reduction(+:wave_function_normalisation)
do i=1,n_mc_points
  call random_number(x)
  call random_number(y)
  call random_number(z)
  x = lx*x +integral_bounds(1,1)
  y = ly*y +integral_bounds(2,1)
   z = lz*z +integral_bounds(3,1)
   position_total(1:3) = [x,y,z]
   if (n_space_dims == 6) then
     call random_number(x)
     call random_number(y)
     call random_number(z)
     x = lx*x +integral_bounds(1,1)
     y = ly*y +integral_bounds(2,1)
     z = lz*z +integral_bounds(3,1)
     position_total(4:6) = [x,y,z]
   end if
   wave_function_normalisation = wave_function_normalisation&
     +wave_function(position_total,dof_coefficients)**2
 end do
 wave_function_normalisation = lx*ly*lz*wave_function_normalisation/n_mc_points
 if (n_space_dims == 6) then
   wave_function_normalisation = lx*ly*lz*wave_function_normalisation
 end if
 if (allocated(seed)) deallocate(seed)
 deallocate(position_total)
```

References basis_functions::n_space_dims, and basis_functions::wave_function.

Referenced by calculations::calc().

Here is the caller graph for this function:



11.6 energy_plotting Namespace Reference

Script outputs the energy against the bond length.

Variables

- data = np.loadtxt('energies.txt')

  Loading in the data from the energies.txt.
- bond_length = data[:,0]
- energy = data[:,1]

11.6.1 Detailed Description

Script outputs the energy against the bond length.

11.6.2 Variable Documentation

11.6.2.1 bond_length

```
energy_plotting.bond_length = data[:,0]
```

Parameters

bond_length	contains all the bond lengths evaluated

Definition at line 14 of file energy_plotting.py.

11.6.2.2 data

```
energy_plotting.data = np.loadtxt('energies.txt')
```

Loading in the data from the energies.txt.

Parameters

data	contains the bond lengths and energy conatined in the energies.txt file

Definition at line 10 of file energy_plotting.py.

11.6.2.3 energy

```
energy_plotting.energy = data[:,1]
```

Parameters

energy	conatins the energy results evaluated at their corresponding bond_lengths

Definition at line 16 of file energy_plotting.py.

11.7 gp_surrogate Module Reference

Gaussian process surrogate submodules and functions.

Data Types

- interface cov_kernal_dx_1_interface
- interface cov_kernal_interface
- interface cov_kernal_xx_dx_interface
- interface gp_init

  general initialisation function, see gp_init_gausscov
- interface gp_k_post

  function for postieror covairance kernal use in format x_1, x_2, x1_dim, x2_dim for cov(x_1,x_2) use in format x, x_dim for cov(x,x)
- interface mean_func_interface
- interface mean_func_interface_dx

Functions/Subroutines

- subroutine gp_init_gausscov (mean_prior, mean_prior_dx, ker_var, ker_length, param_data_in, E_data_in, n_data_in, n_dof_in, n_threads_in)

  intialises with a gaussian covariance
- subroutine gp_init_arbcov (mean_prior, cov_prior, param_data_in, E_data_in, n_data_in, n_dof_in)

  do not use, don't have a function for the dervatives of the kernal/mean
- real(dp) function, dimension(x_dim) gp_mu_post (x, x_dim)

  posterior mean

- real(dp) function gp_mu_post_dx (x, dim)

  derivative of the posterior mean, wrt dimension dim
- real(dp) function, dimension(x_dim, n_dof) gp_mu_post_grad (x, x_dim, only)

  derivative of the posterior mean, needed for stoch_grad
- real(dp) function, dimension(x_dim, x_dim) gp_k_post_same_x (x, x_dim)

  specific function for cov(x,x), see gp_k_post
- real(dp) function, dimension(x1_dim, x2_dim) gp_k_post_diff_x (x1, x2, x1_dim, x2_dim)

  specific function for cov(x_1,x_2), see gp_k_post
- subroutine gp_post_k_grad (X, x_dim, out)

  grad of the posterior covariance, evaluated at X of size x_dim,n_dof, needed for stoch_grad
- subroutine gp_update (param_data_in_top, E_data_in_top, n_top, Algo_choice)

  update routine for adding data to gp
- subroutine gp_debug_print_state ()

  prints all scalar stae variables, size of all array state variables, and min and max stored energy
- subroutine gp_exit ()

  deallocates state variables
- subroutine gp_min_stored (min_params, min_E)

  for finding minimum of the energy and associated parameter space point, from stored data
- subroutine gp_return_size_data (n_data_out, n_dof_out)

  returns the integers that control the size of the state variables
- subroutine gp_return_state_data (kernel_var_out, kernal_inv_length_out, param_data_out, E_data_out, param_pres_out, param_cov_out, data_mean_out)

  for acessing a copy of the state variables, they are returned to the intent(out) parameter with the corresponding name
- subroutine gp_restart (n_data_in, n_dof_in, kernel_var_in, kernal_inv_length_in, param_data_in, E_data_in, param_pres_in, param_cov_in, data_mean_in, mean_prior, mean_prior_dx, n_threads_in)

  sets a state from inputed data, intended for use with restart files.

Variables

- procedure(mean_func_interface_dx), pointer mu_prior_dx => null()

11.7.1 Detailed Description

Gaussian process surrogate submodules and functions.

11.7.2 Function/Subroutine Documentation

11.7.2.1 gp_debug_print_state()

```
subroutine gp_surrogate::gp_debug_print_state
```

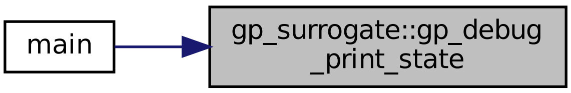
prints all scalar stae variables, size of all array state variables, and min and max stored energy

Definition at line 639 of file GP_surrogate.f90.

```
    implicit none

    print*, "init=", init
    print*, "n_data=", n_data
    print*, "n_dof=", n_dof
    print*, "kernel_var=", kernel_var
    print*, "kernal_inv_length=", kernal_inv_length
    print*, "shape of param_data=", shape(param_data)
    print*, "shape of E_data=", shape(e_data)
    print*, "shape of param_pres=", shape(param_pres)
    print*, "shape of param_cov=", shape(param_cov)
    print*, "shape of data_mean=", shape(data_mean)
    print*, "min energy", minval(e_data)
    print*, "max energy", maxval(e_data)

```

Referenced by main().

Here is the caller graph for this function:



11.7.2.2 gp_exit()

```
subroutine gp_surrogate::gp_exit
```

deallocates state variables

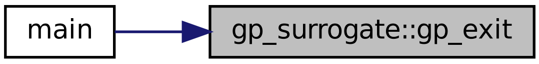
Definition at line 658 of file GP_surrogate.f90.

```
    implicit none

        !deallocating state variables
        deallocate(param_data,e_data,param_pres, param_cov,data_mean)

        !deallocating pointers
        nullify(mu_prior,mu_prior_dx ,k_prior ,k_prior_dx_1_i ,k_prior_xx_dx_i ,gp_k_post_xx_d_x_i)

```

References mu_prior_dx.

Referenced by main().

Here is the caller graph for this function:



11.7.2.3 gp_init_arbcov()

```
subroutine gp_surrogate::gp_init_arbcov (
              procedure(mean_func_interface) mean_prior,
              procedure(cov_kernal_interface) cov_prior,
              real(dp), dimension(n_data_in,n_dof_in), intent(in) param_data_in,
              real(dp), dimension(n_data_in), intent(in) E_data_in,
              integer, intent(in) n_data_in,
              integer, intent(in) n_dof_in )
```

do not use, don't have a function for the dervatives of the kernal/mean

Definition at line 194 of file GP_surrogate.f90.

```
        implicit none
        procedure(mean_func_interface) :: mean_prior
        procedure(cov_kernal_interface) :: cov_prior
        integer, intent(in) :: n_data_in, n_dof_in
        real(dp), dimension(n_data_in,n_dof_in), intent(in) :: param_data_in
        real(dp), dimension(n_data_in), intent(in) :: E_data_in
        !work and ipiv are needed for lapack call, have no useful info
        integer :: i,j

        !setting priors
        mu_prior => mean_prior
        k_prior => cov_prior

        !setting data
        n_data = n_data_in
        n_dof = n_dof_in
        allocate(param_data(n_data,n_dof))
        allocate(e_data(n_data))
        e_data = e_data_in
        param_data = param_data_in


        allocate(param_cov(n_data,n_data))
        allocate(param_pres(n_data,n_data))

        !finds covarience for new data
        do i=1,n_data
            do j=1,i
                !is symmteric, this reduces calls
                param_cov(i,j) = k_prior(param_data(i,:), param_data(j,:))
                param_cov(j,i) = param_cov(i,j)
            end do
        end do

        !compute inverse of cov (precsion)
        param_pres = param_cov
        call svd_inverse(param_pres,  n_data)

        allocate(data_mean(n_data))
        !updating the prior mean list
        do i=1,n_data
            data_mean(i) = mu_prior(param_data(i,:))
        end do

        init = .true.

```

11.7.2.4 gp_init_gausscov()

```
subroutine gp_surrogate::gp_init_gausscov (
              procedure(mean_func_interface) mean_prior,
              procedure(mean_func_interface_dx) mean_prior_dx,
              real(dp), intent(in) ker_var,
              real(dp), intent(in) ker_length,
              real(dp), dimension(n_data_in,n_dof_in), intent(in) param_data_in,
              real(dp), dimension(n_data_in), intent(in) E_data_in,
              integer, intent(in) n_data_in,
              integer, intent(in) n_dof_in,
              integer, intent(in) n_threads_in )
```

intialises with a gaussian covariance

Definition at line 133 of file GP_surrogate.f90.

```
    implicit none
    procedure(mean_func_interface) :: mean_prior
    procedure(mean_func_interface_dx) :: mean_prior_dx
    integer, intent(in) :: n_data_in, n_dof_in,n_threads_in
    real(dp), intent(in) :: ker_var, ker_length
    real(dp), dimension(n_data_in,n_dof_in), intent(in) :: param_data_in
    real(dp), dimension(n_data_in), intent(in) :: E_data_in
    integer :: i,j

    n_threads=n_threads_in

    !setting prior parameters
    kernel_var = ker_var
    kernal_inv_length = 1.0_dp/ker_length
    !setting priors
    mu_prior => mean_prior
    k_prior => gaussian_kernel

    !derivatives of priors, needed for SGA step later
    mu_prior_dx => mean_prior_dx
    k_prior_dx_1_i => gaussian_kernel_dx_1_i
    k_prior_xx_dx_i => zero_func
    gp_k_post_xx_d_x_i => zero_func !prior 0=>post 0, this saves having to evaluate (or write, until
 support is expanded) code to get the 0

    !setting data
    n_data = n_data_in
    n_dof = n_dof_in
    allocate(param_data(n_data,n_dof))
    allocate(e_data(n_data))
    e_data = e_data_in
    param_data = param_data_in


    allocate(param_cov(n_data,n_data))
    allocate(param_pres(n_data,n_data))

    !finds covarience for new data
    do i=1,n_data
        do j=1,i
            !is symmteric, this reduces calls
            param_cov(i,j) = k_prior(param_data(i,:), param_data(j,:))
            param_cov(j,i) = param_cov(i,j)
        end do
    end do

    !compute inverse of cov (precsion)
    param_pres = param_cov
    call svd_inverse(param_pres,  n_data)

    allocate(data_mean(n_data))
    !updating the prior mean list
    do i=1,n_data
        data_mean(i) = mu_prior(param_data(i,:))
    end do

    init = .true.

```

References mu_prior_dx.

11.7.2.5 gp_k_post_diff_x()

```
real(dp) function, dimension(x1_dim,x2_dim) gp_surrogate::gp_k_post_diff_x (
              real(dp), dimension(x1_dim,n_dof), intent(in) x1,
              real(dp), dimension(x2_dim,n_dof), intent(in) x2,
              integer, intent(in) x1_dim,
              integer, intent(in) x2_dim )
```

specific function for cov(x_1,x_2), see gp_k_post

Definition at line 356 of file GP_surrogate.f90.

```
        implicit none
        integer, intent(in) :: x1_dim, x2_dim
        real(dp), dimension(x1_dim,n_dof), intent(in) :: x1 !param space point
        real(dp), dimension(x2_dim,n_dof), intent(in) :: x2 !param space point
        real(dp), dimension(x1_dim,x2_dim) :: out, x1_cov_x2
        real(dp), dimension(n_data,x2_dim) :: data_cov_x2
        real(dp), dimension(x1_dim,n_data) :: x1_cov_data
        integer :: i,j

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        do i=1,x2_dim
            do j=1,n_data
                data_cov_x2(j,i) = k_prior(x2(i,:), param_data(j,:))
            end do
        end do

        do i=1,x1_dim
            do j=1,n_data
                x1_cov_data(i,j) = k_prior(param_data(j,:),x1(i,:))
            end do
        end do

        do i=1,x1_dim
            do j=1,x2_dim
                x1_cov_x2(i,j) = k_prior(x1(i,:), x2(j,:))
            end do
        end do

       out = x1_cov_x2 + matmul(matmul(x1_cov_data,param_pres),data_cov_x2)

```

11.7.2.6 gp_k_post_same_x()

```
real(dp) function, dimension(x_dim,x_dim) gp_surrogate::gp_k_post_same_x (
              real(dp), dimension(x_dim,n_dof), intent(in) x,
              integer, intent(in) x_dim )
```

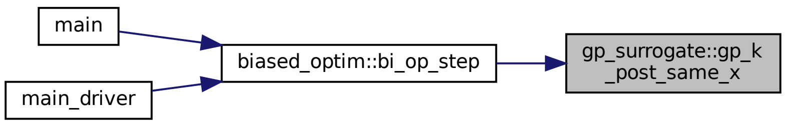
specific function for cov(x,x), see gp_k_post

Definition at line 322 of file GP_surrogate.f90.

```
        implicit none
        integer, intent(in) :: x_dim
        real(dp), dimension(x_dim,n_dof), intent(in) :: x !param space point
        real(dp), dimension(x_dim,x_dim) :: out, x_cov_x
        real(dp), dimension(x_dim,n_data) :: x_cov_data
        real(dp), dimension(n_data,x_dim) :: data_cov_x
        integer :: i,j

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        do i=1,x_dim
            do j=1,n_data
                data_cov_x(j,i) = k_prior(param_data(j,:), x(i,:))
            end do
        end do

        x_cov_data = transpose(data_cov_x)
        do i=1,x_dim
            do j=1,i
                !is symmteric, this reduces calls
                x_cov_x(i,j) = k_prior(x(i,:), x(j,:))
                x_cov_x(j,i) = x_cov_x(i,j)
            end do
        end do

       out = x_cov_x + matmul(matmul(x_cov_data,param_pres),data_cov_x)

```

Referenced by biased_optim::bi_op_step().

Here is the caller graph for this function:



11.7.2.7 gp_min_stored()

```
subroutine gp_surrogate::gp_min_stored (
             real(dp), dimension(n_dof), intent(out) min_params,
             real(dp), intent(out) min_E )
```

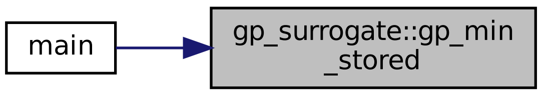
for finding minimum of the energy and associated parameter space point, from stored data

Definition at line 724 of file GP_surrogate.f90.

```
        implicit none
        real(dp), dimension(n_dof), intent(out) :: min_params
        real(dp), intent(out) :: min_E
        integer :: minloc_E

        minloc_e = minloc(e_data,1)

        min_e = e_data(minloc_e)
        min_params = param_data(minloc_e,:)

```

Referenced by main().

Here is the caller graph for this function:



11.7.2.8 gp_mu_post()

```
real(dp) function, dimension(x_dim) gp_surrogate::gp_mu_post (
            real(dp), dimension(x_dim,n_dof), intent(in) x,
            integer, intent(in) x_dim )
```

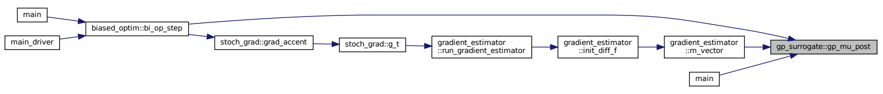
posterior mean

Definition at line 244 of file GP_surrogate.f90.

```
        implicit none
        integer, intent(in) :: x_dim
        real(dp), dimension(x_dim,n_dof), intent(in) :: x
        real(dp), dimension(x_dim) :: out
        real(dp), dimension(x_dim) :: x_means
        real(dp), dimension(x_dim,n_data) :: x_cov_data
        integer :: i,j

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        do i=1,x_dim
            x_means(i) = mu_prior(x(i,:))
            do j=1,n_data
                x_cov_data(i,j) = k_prior(x(i,:), param_data(j,:))
            end do
        end do

       out = x_means + matmul(matmul(x_cov_data,param_pres),(e_data-data_mean))

```

Referenced by biased_optim::bi_op_step(), gradient_estimator::m_vector(), and main().

Here is the caller graph for this function:



11.7.2.9 gp_mu_post_dx()

```
real(dp) function gp_surrogate::gp_mu_post_dx (
            real(dp), dimension(n_dof), intent(in) x,
            integer, intent(in) dim )
```

derivative of the posterior mean, wrt dimension dim

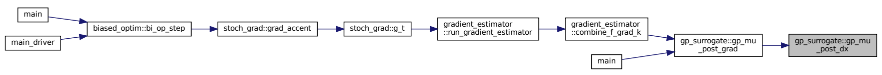
Definition at line 270 of file GP_surrogate.f90.

```
        implicit none
        !only gets evaluated at one point
        integer, intent(in) :: dim
        real(dp), dimension(n_dof), intent(in) :: x
        real(dp) :: out
        real(dp), dimension(1,n_data) :: x_cov_data_dx
        real(dp), dimension(1) :: prod_temp
        integer :: j

        do j=1,n_data
            x_cov_data_dx(1,j) = k_prior_dx_1_i(x,param_data(j,:),dim)
        end do

        prod_temp = matmul(x_cov_data_dx,matmul(param_pres,(e_data-data_mean)))
        out = mu_prior_dx(x, dim) + prod_temp(1)

```

References mu_prior_dx.

Referenced by gp_mu_post_grad().

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::combine_f_grad_k
main
gp_surrogate::gp_mu _post_grad
gp_surrogate::gp_mu _post_dx

11.7.2.10 gp_mu_post_grad()

```
real(dp) function, dimension(x_dim,n_dof) gp_surrogate::gp_mu_post_grad (
            real(dp), dimension(x_dim,n_dof), intent(in) x,
            integer, intent(in) x_dim,
            integer, intent(in), optional only )
```

derivative of the posterior mean, needed for stoch_grad

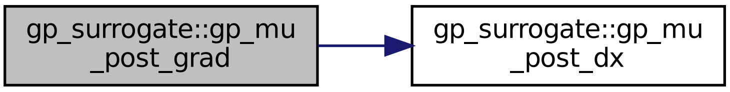
Definition at line 291 of file GP_surrogate.f90.

```
        implicit none
        !row i contain the gradient of mu wrt data_i
        integer, intent(in) :: x_dim
        integer, intent(in), optional :: only
        real(dp), dimension(x_dim,n_dof), intent(in) :: x !param space point
        real(dp), dimension(x_dim,n_dof) :: out
        integer :: i,j

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        out=0

        if (present(only)) then
            do j=1,n_dof
                out(only,j) = gp_mu_post_dx(x(only,:),j)
            end do
        else
            do i=1,x_dim
                do j=1,n_dof
                    out(i,j) = gp_mu_post_dx(x(i,:),j)
                end do
            end do
        end if

```

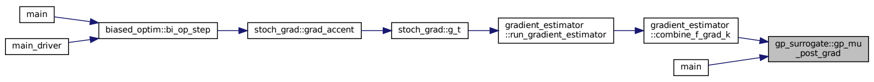
References gp_mu_post_dx().

Referenced by gradient_estimator::combine_f_grad_k(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.2.11 gp_post_k_grad()

```
subroutine gp_surrogate::gp_post_k_grad (
            real(dp), dimension(x_dim,n_dof), intent(in) X,
            integer, intent(in) x_dim,
            real(dp), dimension(x_dim,x_dim,x_dim,n_dof), intent(out) out )
```

grad of the posterior covariance, evaluated at X of size x_dim,n_dof, needed for stoch_grad

Parameters

in	*x_dim*	size of X
in	*x*	point to evaluate at
out	*out*	the output matrix, size (x_dim,x_dim,x_dim,n_dof)

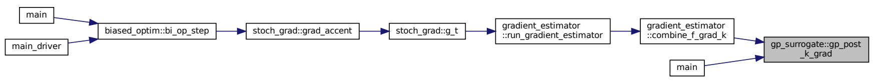
Definition at line 422 of file GP_surrogate.f90.

```
        !triple nested for loops are slow,try not to call often
        implicit none
        integer,intent(in) :: x_dim
        real(dp), dimension(x_dim,n_dof), intent(in) :: X
        real(dp), dimension(x_dim,x_dim,x_dim,n_dof),intent(out) ::  out
        integer :: i,j,l

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        do i=1,x_dim
            do j=1,x_dim
                if (i/=j) then
                    do l=1,n_dof
                        out(i,j,j,l) = gp_k_post_d_x_1_i(x(j,:),x(i,:),l)
                        out(i,j,i,l) = gp_k_post_d_x_1_i(x(i,:),x(j,:),l)
                    end do
                else
```

```
442                     do l=1,n_dof
443                         out(i,j,i,l) = gp_k_post_xx_d_x_i(x(i,:),l)
444                     end do
445                 end if
446             end do
447         end do
448
```

Referenced by gradient_estimator::combine_f_grad_k(), and main().

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::combine_f_grad_k
main
gp_surrogate::gp_post _k_grad

11.7.2.12 gp_restart()

```
subroutine gp_surrogate::gp_restart (
              integer, intent(in) n_data_in,
              integer, intent(in) n_dof_in,
              real(dp), intent(in) kernel_var_in,
              real(dp), intent(in) kernal_inv_length_in,
              real(dp), dimension(n_data_in,n_dof_in), intent(in) param_data_in,
              real(dp), dimension(n_data_in), intent(in) E_data_in,
              real(dp), dimension(n_data_in,n_data_in), intent(in) param_pres_in,
              real(dp), dimension(n_data_in,n_data_in), intent(in) param_cov_in,
              real(dp), dimension(n_data_in), intent(in) data_mean_in,
              procedure(mean_func_interface) mean_prior,
              procedure(mean_func_interface_dx) mean_prior_dx,
              integer, intent(in) n_threads_in )
```

sets a state from inputed data, intended for use with restart files.

Variables must correspond to named counterpart

Definition at line 770 of file GP_surrogate.f90.

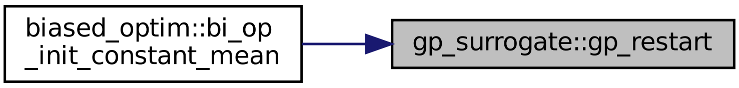
```
770         implicit none
771         integer, intent(in) :: n_data_in, n_dof_in,n_threads_in
772         real(dp), intent(in) :: kernel_var_in, kernal_inv_length_in
773         real(dp), dimension(n_data_in,n_dof_in), intent(in):: param_data_in
774         real(dp), dimension(n_data_in), intent(in) :: E_data_in
775         real(dp), dimension(n_data_in,n_data_in), intent(in) :: param_pres_in, param_cov_in
776         real(dp), dimension(n_data_in), intent(in) :: data_mean_in
777         procedure(mean_func_interface) :: mean_prior
778         procedure(mean_func_interface_dx) :: mean_prior_dx
779
780         n_data = n_data_in
781         n_dof = n_dof_in
782         n_threads = n_threads_in
783
784         kernel_var=kernel_var_in
785         kernal_inv_length=kernal_inv_length_in
786         allocate(param_data(n_data,n_dof))
787         param_data=param_data_in
788         allocate(e_data(n_data))
789         e_data=e_data_in
790         allocate(param_pres(n_data,n_data))
791         param_pres=param_pres_in
792         allocate(param_cov(n_data,n_data))
793         param_cov=param_cov_in
794         allocate(data_mean(n_data))
```

```
        data_mean=data_mean_in


        mu_prior => mean_prior
        k_prior => gaussian_kernel

        !derivatives of priors, needed for SGA step later
        mu_prior_dx => mean_prior_dx
        k_prior_dx_1_i => gaussian_kernel_dx_1_i
        k_prior_xx_dx_i => zero_func
        gp_k_post_xx_d_x_i => zero_func

        init = .true.

```

References mu_prior_dx.

Referenced by biased_optim::bi_op_init_constant_mean().

Here is the caller graph for this function:



11.7.2.13 gp_return_size_data()

```
subroutine gp_surrogate::gp_return_size_data (
             integer, intent(out) n_data_out,
             integer, intent(out) n_dof_out )
```

returns the integers that control the size of the state variables

Definition at line 738 of file GP_surrogate.f90.

```
    implicit none
    integer,intent(out) :: n_data_out,  n_dof_out

    n_data_out = n_data
    n_dof_out = n_dof

```

11.7.2.14 gp_return_state_data()

```
subroutine gp_surrogate::gp_return_state_data (
             real(dp), intent(out) kernel_var_out,
             real(dp), intent(out) kernal_inv_length_out,
             real(dp), dimension(n_data,n_dof), intent(out) param_data_out,
             real(dp), dimension(n_data), intent(out) E_data_out,
             real(dp), dimension(n_data,n_data), intent(out) param_pres_out,
             real(dp), dimension(n_data,n_data), intent(out) param_cov_out,
             real(dp), dimension(n_data), intent(out) data_mean_out )
```

for acessing a copy of the state variables, they are returned to the intent(out) parameter with the corresponding name

Definition at line 749 of file GP_surrogate.f90.

```
      implicit none
      real(dp), intent(out)  :: kernel_var_out, kernal_inv_length_out
      real(dp), dimension(n_data,n_dof), intent(out):: param_data_out
      real(dp), dimension(n_data), intent(out) :: E_data_out
      real(dp), dimension(n_data,n_data), intent(out) :: param_pres_out, param_cov_out
      real(dp), dimension(n_data), intent(out) :: data_mean_out

      kernel_var_out = kernel_var
      kernal_inv_length_out = kernal_inv_length
      param_data_out = param_data
      e_data_out = e_data
      param_pres_out = param_pres
      param_cov_out = param_cov
      data_mean_out = data_mean

```

11.7.2.15 gp_update()

```
subroutine gp_surrogate::gp_update (
            real(dp), dimension(n_top,n_dof), intent(in) param_data_in_top,
            real(dp), dimension(n_top), intent(in) E_data_in_top,
            integer, intent(in) n_top,
            character(len=1), intent(in) Algo_choice )
```

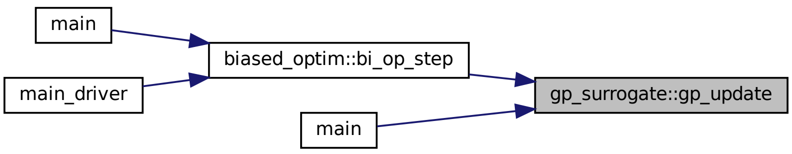
update routine for adding data to gp

Definition at line 453 of file GP_surrogate.f90.

```
      integer, intent(in) :: n_top
      real(dp), dimension(n_top,n_dof), intent(in) :: param_data_in_top
      real(dp), dimension(n_top), intent(in) :: E_data_in_top
      character(len=1), intent(in) :: Algo_choice

      select case (algo_choice)
      case ("F")
         call gp_update_full_svd(param_data_in_top, e_data_in_top, n_top)
      case ("U")
         call gp_update_woodbury_block_update(param_data_in_top, e_data_in_top, n_top)
      case default
         print*, "please make a valid algorithm choice"
      end select

```

Referenced by biased_optim::bi_op_step(), and main().

Here is the caller graph for this function:



11.7.3 Variable Documentation

11.7.3.1 mu_prior_dx

```
procedure (mean_func_interface_dx), pointer gp_surrogate::mu_prior_dx => null()
```

Definition at line 18 of file GP_surrogate.f90.

```
18     procedure(mean_func_interface_dx),pointer :: mu_prior_dx => null()
```

Referenced by gp_exit(), gp_init_gausscov(), gp_mu_post_dx(), and gp_restart().

11.8 gradient_estimator Module Reference

Subroutines for obtaining a gradient estimation through Cholesky decomposition.

Functions/Subroutines

- subroutine get_cholesky (cholesky_decomp, matrix_in)

  performs Cholesky decomposition without destroying the old array.
- real(dp) function, dimension(:), allocatable m_vector (x, previous_best)

  builds the "m_vector" as described in: arXiv:1602.05149v4 [stat.ML] 5 May 2019 contains the difference between the mean and best result in a vector.
- real(dp) function, dimension(:), allocatable z_vector (z_in)

  build the "z_vector" as described in: arXiv:1602.05149v4 [stat.ML] 5 May 2019
- integer function, dimension(1) find_max_thread (m_vec, z_vec, c_mat)

  finds the location in the vector that corresponds to the maximum of "m + CZ"
- subroutine check_max_is_unique (m_vec, z_vec, c_mat, find_max_thread)

  check if the maximum found in the find_max_thread is a unique value.
- subroutine init_diff_f (x, z_in, previous_best)

  initialises the workspace F and Cholesky decomposition of the covariance matrix.
- subroutine bkwd_diff_f ()

  performs the backward difference differentiation on the workspace F
- subroutine combine_f_grad_k (x, n_params)

  combines the gradient contributions from the workspace F and those associated with the m_vector and covariance matrix.
- real(dp) function, dimension(:,:), allocatable run_gradient_estimator (x, z_in, n_params, previous_best)

  runs the gradient estimator associated routines from one block.

Variables

- real(dp), dimension(:,:), allocatable f_ij
- real(dp), dimension(:,:), allocatable l_ij
- real(dp), dimension(:,:), allocatable gradient_matrix
- real(dp) check_max

11.8.1 Detailed Description

Subroutines for obtaining a gradient estimation through Cholesky decomposition.

11.8.2 Function/Subroutine Documentation

11.8.2.1 bkwd_diff_f()

```
subroutine gradient_estimator::bkwd_diff_f
```

performs the backward difference differentiation on the workspace F

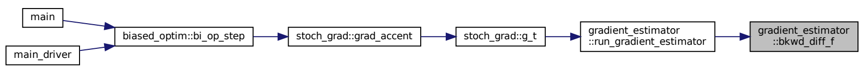
Definition at line 158 of file gradient_estimator.f90.

```
158   integer :: i, j, k, N
159   integer, dimension(2) :: shape_arr
160   shape_arr = shape(f_ij)
161   n = shape_arr(1)
162
163   do k=n,1, -1
164     if (abs(l_ij(k,k)) > 0.0_dp) then
165     !row operations
166     do j = k+1, n
167       do i = j, n
168         f_ij(i,k) = f_ij(i,k) - f_ij(i,j)*l_ij(j,k)
169         f_ij(j,k) = f_ij(j,k) - f_ij(i,j)*l_ij(i,k)
170       end do
171     end do
172     !lead column
173     do j = k+1, n
174       f_ij(j,k) = f_ij(j,k)/l_ij(k,k)
175       f_ij(k,k) = f_ij(k,k) - l_ij(j,k)* f_ij(j,k)
176     end do
177     !pivot
178     f_ij(k,k) = 0.5_dp*f_ij(k,k)/l_ij(k,k)
179     end if
180   end do
```

References f_ij, and l_ij.

Referenced by run_gradient_estimator().

Here is the caller graph for this function:



11.8.2.2 check_max_is_unique()

```
subroutine gradient_estimator::check_max_is_unique (
            real(dp), dimension(:), intent(in) m_vec,
            real(dp), dimension(:), intent(in) z_vec,
            real(dp), dimension(:,:), intent(in) c_mat,
            integer, dimension(1), intent(in) find_max_thread )
```

check if the maximum found in the find_max_thread is a unique value.

Parameters

m_vec	as returned from the m_vector function.
z_vec	as returned from the z_vector function.
c_mat	corresponds to the Cholesky decomposition of the covariance matrix.
find_max_thread	integer containing the location of the maximum of "m + CZ"

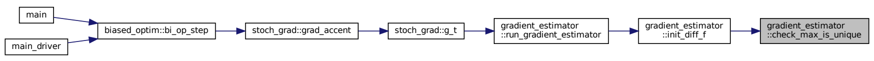
Definition at line 87 of file gradient_estimator.f90.

```
    integer, dimension(1), intent(in) :: find_max_thread
    real(dp), dimension(:,:) , intent(in):: c_mat
    real(dp), dimension(:), intent(in) :: m_vec, z_vec
    real(dp), dimension(:), allocatable :: f_vector
    integer :: i
    !make sure indexing starts from 0, so is consistent.
    allocate(f_vector(0:( size(z_vec) -1 )))
    !by default set = 1, so it does nothing when multiplied with something.
    check_max = 1.0_dp
    f_vector = m_vec + matmul(c_mat, z_vec)
    !loop over each element of the calculated vector and check if it equals the max of that vector
    !but in more than one location.
    do i = 0, size(f_vector) -1
       if ( (f_vector(i) .EQ. f_vector(find_max_thread(1))) .AND. (i .NE. find_max_thread(1)) ) then
         check_max=0.0_dp
       end if
     end do
     deallocate(f_vector)
```

References check_max.

Referenced by init_diff_f().

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::init_diff_f
gradient_estimator ::check_max_is_unique

11.8.2.3 combine_f_grad_k()

```
subroutine gradient_estimator::combine_f_grad_k (
              real(dp), dimension(:,:)  x,
              integer n_params )
```

combines the gradient contributions from the workspace F and those associated with the m_vector and covariance matrix.

Parameters

x	is 2D array containing the param_inputs for each thread.
n_params	is the total number of parameters being evaluated

Definition at line 189 of file gradient_estimator.f90.

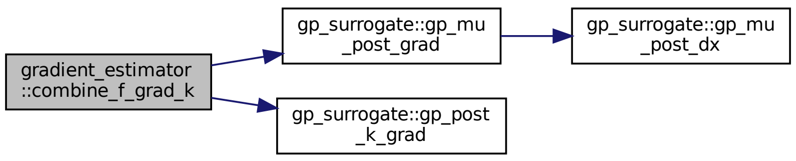
```
  real(dp), dimension(:,:) :: x
  real(dp), dimension(:,:), allocatable :: grad_mu
  real(dp), dimension(:,:,:,:), allocatable:: post_grad_in
```

```
integer, dimension(2) :: shape_arr
integer, dimension(4) :: shape_arr_4
integer :: i, j, k, l, n_params


shape_arr = shape(x)

if allocated(post_grad_in) then
  deallocate(post_grad_in)
end if
allocate(post_grad_in(shape_arr(1), shape_arr(1), shape_arr(1), n_params))
if allocated(grad_mu) then
  deallocate(grad_mu)
end if
allocate(grad_mu( shape_arr(1), n_params ))
call gp_post_k_grad(x, shape_arr(1), post_grad_in)
shape_arr_4 = shape(post_grad_in)

!print*, shape_arr_4(:)
if allocated(gradient_matrix) then
  deallocate(gradient_matrix)
end if
allocate(gradient_matrix(shape_arr_4(3), shape_arr_4(4))  )
gradient_matrix = 0.0_dp
do k = 1, shape_arr_4(3)
  do l = 1, shape_arr_4(4)
    do i = 1, shape_arr_4(1)
      do j = 1, shape_arr_4(2)
        gradient_matrix(k,l) = gradient_matrix(k,l) + f_ij(i,j)*post_grad_in(i,j,k,l)
      end do
    end do
  end do
end do
!gradient in param space associated with each mean.
grad_mu = gp_mu_post_grad(x,shape_arr(1))

gradient_matrix = (gradient_matrix - grad_mu)


!deallocate(F_ij, L_ij)
```

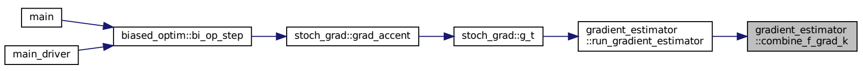
References f_ij, gp_surrogate::gp_mu_post_grad(), gp_surrogate::gp_post_k_grad(), and gradient_matrix.

Referenced by run_gradient_estimator().

Here is the call graph for this function:


gradient_estimator ::combine_f_grad_k
gp_surrogate::gp_mu _post_grad
gp_surrogate::gp_mu _post_dx
gp_surrogate::gp_post _k_grad

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::combine_f_grad_k

11.8.2.4 find_max_thread()

```
integer function, dimension(1) gradient_estimator::find_max_thread (
            real(dp), dimension(:), intent(in) m_vec,
            real(dp), dimension(:), intent(in) z_vec,
            real(dp), dimension(:,:), intent(in) c_mat )
```

finds the location in the vector that corresponds to the maximum of "m + CZ"

Parameters

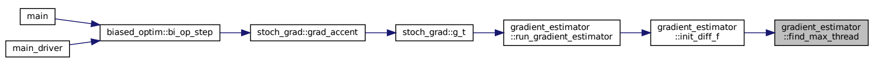
m_vec	as returned from the m_vector function.
z_vec	as returned from the z_vector function.
c_mat	corresponds to the Cholesky decomposition of the covariance matrix.

Definition at line 70 of file gradient_estimator.f90.

```
    real(dp), dimension(:,:) , intent(in):: c_mat
    real(dp), dimension(:), intent(in) :: m_vec, z_vec
    integer, dimension(1) :: find_max_thread
    integer, dimension(2) :: shape_arr
    shape_arr = shape(c_mat)
    find_max_thread = maxloc(m_vec + matmul(c_mat, z_vec)) - 1
    !-1 is shift the indexing back to 0 from 1.
```

Referenced by init_diff_f().

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::init_diff_f
gradient_estimator ::find_max_thread

11.8.2.5 get_cholesky()

```
subroutine gradient_estimator::get_cholesky (
            real(dp), dimension(:,:)  cholesky_decomp,
            real(dp), dimension(:,:)  matrix_in )
```

performs Cholesky decomposition without destroying the old array.

also with 0 at i=0 padding.

Parameters

cholesky_decomp	is where the result is stored.
matrix_in	2D matrix to be decomposed.

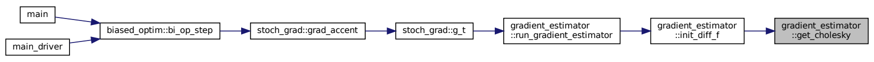
Definition at line 20 of file gradient_estimator.f90.

```
  real(dp), dimension(:,:) :: matrix_in, cholesky_decomp
  real(dp), dimension(:,:), allocatable :: get_cholesky_temp
  integer :: info
```

```
integer, dimension(2) :: shape_arr
get_cholesky_temp = matrix_in
shape_arr = shape(matrix_in)
call dpotrf( 'L', shape_arr(1), get_cholesky_temp, shape_arr(1),info) !this is the LAPACK routine for
  getting Cholesky 'L' = lower triangle. info has an error state
cholesky_decomp = 0.0_dp
cholesky_decomp(1:shape_arr(1), 1:shape_arr(1)) = -get_cholesky_temp
```

Referenced by init_diff_f().

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator::run_gradient_estimator
gradient_estimator::init_diff_f
gradient_estimator::get_cholesky

11.8.2.6 init_diff_f()

```
subroutine gradient_estimator::init_diff_f (
              real(dp), dimension(:,:), intent(in) x,
              real(dp), dimension(:), intent(in) z_in,
              real(dp), intent(in) previous_best )
```

initialises the workspace F and Cholesky decomposition of the covariance matrix.

Parameters

x	is 2D array containing the param_inputs for each thread.
z_in	is a 1D array of normally distributed numbers.
previous_best	is a real number containing the best energy from the prior trial.

Definition at line 113 of file gradient_estimator.f90.

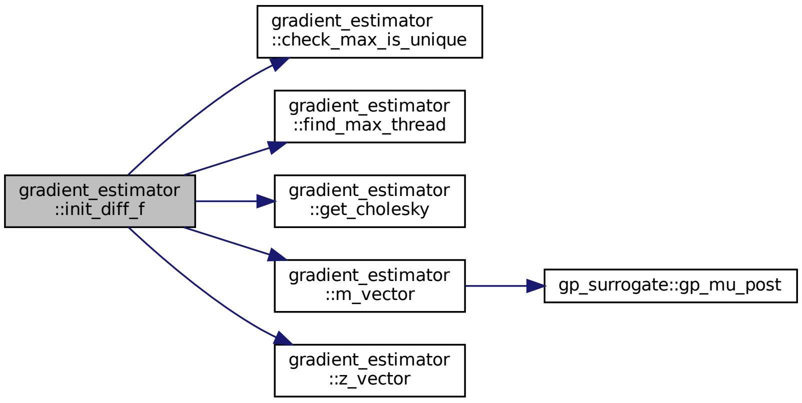
```
real(dp), dimension(:,:) , intent(in):: x
real(dp), intent(in) :: previous_best
real(dp), dimension(:), intent(in) :: z_in
real(dp), dimension(:), allocatable :: m_vec, z_vec
real(dp), dimension(:,:), allocatable :: c_mat, init_F_ij
integer, dimension(2) :: shape_arr
integer, dimension(1) :: loc_max_thread
integer :: i, j
shape_arr = shape(x)
allocate(m_vec(0:shape_arr(1)))
allocate(z_vec(0:shape_arr(1)))
allocate(c_mat(0:shape_arr(1), 0:shape_arr(1)  ))
allocate(init_f_ij(0:shape_arr(1), 0:shape_arr(1)))

if (allocated(f_ij)) then
  deallocate(f_ij)
end if
allocate(f_ij(1:shape_arr(1), 1:shape_arr(1)))

if (allocated(l_ij)) then
  deallocate(l_ij)
end if
allocate(l_ij(1:shape_arr(1), 1:shape_arr(1)))
init_f_ij = 0.0_dp
m_vec = m_vector(x, previous_best)
z_vec = z_vector(z_in)

call get_cholesky(c_mat, gp_k_post(x, shape_arr(1))  )
loc_max_thread = find_max_thread(m_vec, z_vec, c_mat)
!print*, loc_max_thread
```

```
!This is the analytical result of diffing the max(f) w.r.t each L(ij)
init_f_ij( loc_max_thread(1) , 1:loc_max_thread(1)  ) = -z_vec(1:loc_max_thread(1))
!putting array back into correct size (removing index 0, padding)
f_ij =   init_f_ij(1:shape_arr(1) , 1:shape_arr(1))
!put cholesky in same size array, removing 0 index padding.
l_ij = c_mat(1:shape_arr(1) , 1:shape_arr(1))


call check_max_is_unique(m_vec, z_vec, c_mat, loc_max_thread)
deallocate(c_mat, z_vec, m_vec, init_f_ij)
```

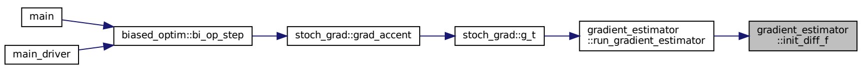
References check_max_is_unique(), f_ij, find_max_thread(), get_cholesky(), l_ij, m_vector(), and z_vector().

Referenced by run_gradient_estimator().

Here is the call graph for this function:


gradient_estimator ::init_diff_f
gradient_estimator ::check_max_is_unique
gradient_estimator ::find_max_thread
gradient_estimator ::get_cholesky
gradient_estimator ::m_vector
gp_surrogate::gp_mu_post
gradient_estimator ::z_vector

Here is the caller graph for this function:



11.8.2.7 m_vector()

```
real(dp) function, dimension(:), allocatable gradient_estimator::m_vector (
            real(dp), dimension(:,:), intent(in) x,
            real(dp), intent(in) previous_best )
```

builds the "m_vector" as described in: arXiv:1602.05149v4 [stat.ML] 5 May 2019 contains the difference between the mean and best result in a vector.

Parameters

x	is 2D array containing the param_inputs for each thread.
previous_best	is a real number containing the best energy from the prior trial.

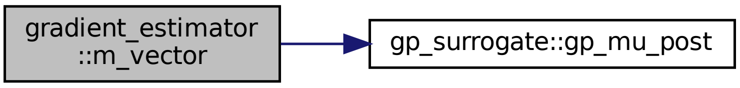
Definition at line 38 of file gradient_estimator.f90.

```
38    real(dp), dimension(:), allocatable :: m_vector, pbest_arr
39    real(dp), dimension(:,:) , intent(in):: x
40    real(dp), intent(in) :: previous_best
41    integer, dimension(2) :: shape_arr
42    shape_arr = shape(x)
43    allocate(m_vector(0:shape_arr(1)))
44    allocate(pbest_arr(shape_arr(1)))
45    pbest_arr = previous_best !need an array to compare each thread/trial/point in param space.
46    m_vector(0) = 0.0_dp
47    m_vector(1:shape_arr(1))=pbest_arr - gp_mu_post(x, shape_arr(1))
```

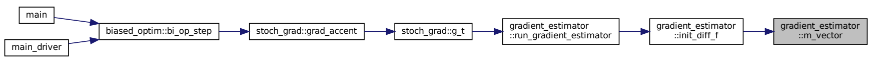
References gp_surrogate::gp_mu_post().

Referenced by init_diff_f().

Here is the call graph for this function:



Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::init_diff_f
gradient_estimator ::m_vector

11.8.2.8 run_gradient_estimator()

```
real(dp) function, dimension(:,:), allocatable gradient_estimator::run_gradient_estimator (
              real(dp), dimension(:,:), intent(in) x,
              real(dp), dimension(:), intent(in) z_in,
              integer, intent(in) n_params,
              real(dp), intent(in) previous_best )
```

runs the gradient estimator associated routines from one block.

Parameters

x	is 2D array containing the param_inputs for each thread.
n_params	is the total number of parameters being evaluated.
z_in	is a 1D array of normally distributed numbers.
previous_best	is a real number containing the best energy from the prior trial.

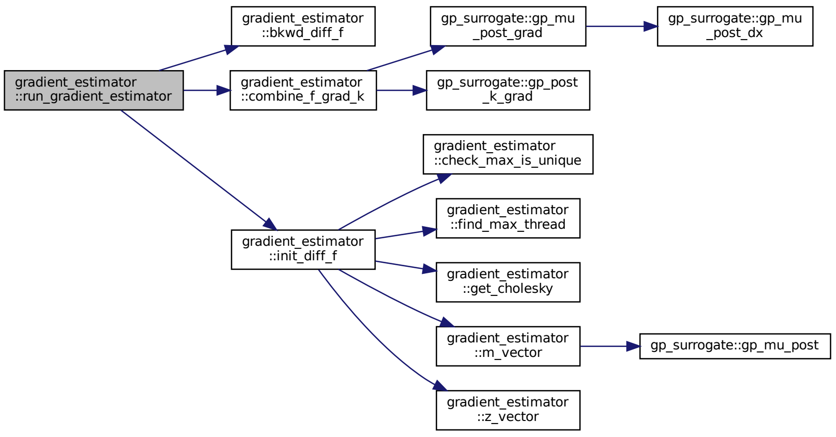
Definition at line 241 of file gradient_estimator.f90.

```
    real(dp), dimension(:,:), intent(in) :: x
    real(dp), dimension(:,:), allocatable :: run_gradient_estimator
    real(dp), dimension(:), intent(in) :: z_in
    integer, intent(in) :: n_params
    real(dp), intent(in) :: previous_best

    call init_diff_f(x,z_in, previous_best)
    call bkwd_diff_f()
    call combine_f_grad_k(x, n_params)
    if allocated(run_gradient_estimator) then
      deallocate(run_gradient_estimator)
    end if
    allocate(run_gradient_estimator, mold=gradient_matrix)
    run_gradient_estimator = gradient_matrix*check_max
    !deallocate(gradient_matrix)
```

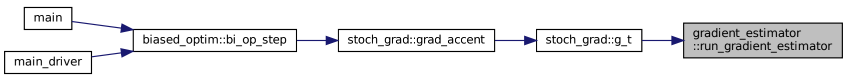
References bkwd_diff_f(), check_max, combine_f_grad_k(), gradient_matrix, and init_diff_f().

Referenced by stoch_grad::g_t().

Here is the call graph for this function:


gradient_estimator ::run_gradient_estimator
gradient_estimator ::bkwd_diff_f
gradient_estimator ::combine_f_grad_k
gradient_estimator ::init_diff_f
gp_surrogate::gp_mu _post_grad
gp_surrogate::gp_mu _post_dx
gp_surrogate::gp_post _k_grad
gradient_estimator ::check_max_is_unique
gradient_estimator ::find_max_thread
gradient_estimator ::get_cholesky
gradient_estimator ::m_vector
gp_surrogate::gp_mu_post
gradient_estimator ::z_vector

Here is the caller graph for this function:



11.8.2.9 z_vector()

```
real(dp) function, dimension(:), allocatable gradient_estimator::z_vector (
              real(dp), dimension(:), intent(in) z_in )
```

build the "z_vector" as described in: arXiv:1602.05149v4 [stat.ML] 5 May 2019

Parameters

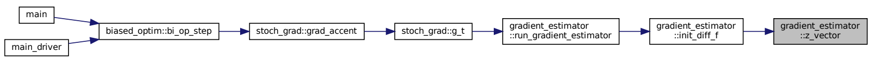
z_in	is a 1D array of normally distributed numbers.

Definition at line 55 of file gradient_estimator.f90.

```
55   real(dp), dimension(:), allocatable :: z_vector
56   real(dp), dimension(:), intent(in) :: z_in
57   integer :: size_z
58   size_z = size(z_in)
59   allocate(z_vector(0:size_z))
60   z_vector(0) = 0
61   z_vector(1:size_z) = z_in
```

Referenced by init_diff_f().

Here is the caller graph for this function:


main
main_driver
biased_optim::bi_op_step
stoch_grad::grad_accent
stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::init_diff_f
gradient_estimator ::z_vector

11.8.3 Variable Documentation

11.8.3.1 check_max

```
real(dp) gradient_estimator::check_max
```

Definition at line 13 of file gradient_estimator.f90.

```
13   real(dp) :: check_max
```

Referenced by check_max_is_unique(), and run_gradient_estimator().

11.8.3.2 f_ij

```
real(dp), dimension(:,:), allocatable gradient_estimator::f_ij
```

Definition at line 12 of file gradient_estimator.f90.

```
12   real(dp), dimension(:,:), allocatable :: F_ij, L_ij, gradient_matrix
```

Referenced by bkwd_diff_f(), combine_f_grad_k(), and init_diff_f().

11.8.3.3 gradient_matrix

```
real(dp), dimension(:,:), allocatable gradient_estimator::gradient_matrix
```

Definition at line 12 of file gradient_estimator.f90.

Referenced by combine_f_grad_k(), and run_gradient_estimator().

11.8.3.4 l_ij

```
real(dp), dimension(:,:), allocatable gradient_estimator::l_ij
```

Definition at line 12 of file gradient_estimator.f90.

Referenced by bkwd_diff_f(), and init_diff_f().

11.9 init_params Namespace Reference

Python functions and scripts to obtain user input import sympy as sym.

Functions

- def get_input (parameter='parameter')
  Function to get a numerical value/parameter through command line.
- def new_param (window, tk_text, tk_entry, tk_input, row, dval='1', text='param', font=font, fs=fontsize)
  Function for creating a row of user entry within the GUI.

Variables

- string title = 'User Input'
- string bg_color = 'white'
- string fg_color = 'grey'
- string bg_button = 'grey'
  note: Mac OS may not support changing button color
- string fg_button = 'black'
- string font = 'Times New Roman'
- int fontsize = 18
- int length = 1000
- int height = 1000
- window = tk.Tk()
- background
- string txt0
- text0 = tk.Label(window,text=txt0,font=(font,fontsize),wraplength=length-5)
  Initializing/Configuring the top row text.
- row
- column
- columnspan

- sticky
- list params
- list def_values = [1,1,1.5,40,1000000,0,10,100,1,10,12345,1,7,0.01,1.0,1.0,5.0,20]
- list floats = [2, 13, 14, 15, 16]
- list integers = [0, 1, 3, 4, 5, 6, 7,8, 9, 10, 11, 12, 17]
- N_params = len(params)
- string txt1 = "The user must provide the following inputs: "
- text1 = tk.Label(window,text=txt1,font=(font,fontsize),wraplength=length-10)
- dval
- text
- string txt2 = "Biased Optimizer Options (not enabled by default):"
- text2 = tk.Label(window,text=txt2,font=(font,fontsize),wraplength=length-10)
- pady
- string txt2_1 = "Enable the biased optimizer?"

    Create a radio button for user to choose between two options.
- text2_1 = tk.Label(window,text=txt2_1,font=(font,fontsize),wraplength=length-10)
- bopt = tk.IntVar()
- variable
- value
- indicatoron
- padx
- entry5 = bopt
- int i = 6
- string txt3 = "Optional user inputs (default options are shown):"
- text3 = tk.Label(window,text=txt3,font=(font,fontsize),wraplength=length-10)
- basis = tk.IntVar()
- entry7 = basis
- string txt4 = "Visualization options:"
- text4 = tk.Label(window,text=txt4,font=(font,fontsize),wraplength=length-10)
- close
- relx
- rely
- anchor
- int p1 = 1

    Checks to see if user input is proper and sets everything to either an integer or float.
- int p2 = 0
- string filename = 'init_params.txt'
- file = open(filename,'w')

11.9.1 Detailed Description

Python functions and scripts to obtain user input import sympy as sym.

11.9.2 Function Documentation

11.9.2.1 get_input()

```
def init_params.get_input (
              parameter = 'parameter' )
```

Function to get a numerical value/parameter through command line.

This is used when a gui cant be used or is unsuitable.
Example code to get the bond length:
x=get_input(parameter='bond length')

Parameters

parameter	The name of the parameter/variable needed from the user.

Definition at line 15 of file init_params.py.

```
def get_input(parameter='parameter'):
    while True:
        x=input('Please enter a value for the {0}: '.format(parameter))
        try:
            x=float(x)
            break
        except:
            print('You did not enter a proper numerical value for the {0}'.format(parameter))
    return x


```

11.9.2.2 new_param()

```
def init_params.new_param (
                window,
                tk_text,
                tk_entry,
                tk_input,
                row,
                dval = '1',
                text = 'param',
                font = font,
                fs = fontsize )
```

Function for creating a row of user entry within the GUI.

Parameters

window	The tkinter window object that has been initialized
tk_text	Variable name (as string) for holding tkinter label
tk_entry	Variable name (as string) for holding tkinter entry object
tk_input	Variable name (as string) for displaying the default parameter value
row	The row of the GUI to display the prompt
dval	Default value of the parameter/variable
text	Name of the parameter/variable
font	The name of the font to be used as given by basis_functions
fontsize	The size of the font

Definition at line 95 of file init_params.py.

```
def new_param(window,
              tk_text,tk_entry,tk_input,
              row,
              dval='1',
              text='param',
               font=font,fs=fontsize):

    # Text to ask user for a specific input
    vars()[tk_text]=tk.Label(window,text=text,font=(font,fs))
    vars()[tk_text].configure(background=bg_color)
    vars()[tk_text].grid(row=row,pady=5,column=0,sticky='e')  #padx=5,

    # Create a space for user text entry
    # Only need first line if you want to show a default value
    vars()[tk_input]=tk.StringVar(value=dval)
    vars()[tk_entry]=tk.Entry(window,width=9,textvariable=vars()[tk_input])
    vars()[tk_entry].grid(column=1,row=row,pady=5,sticky='w')
    return vars()[tk_text],vars()[tk_entry],vars()[tk_input]


```

11.9.3 Variable Documentation

11.9.3.1 anchor

`init_params.anchor`

Definition at line 214 of file init_params.py.

11.9.3.2 background

`init_params.background`

Definition at line 54 of file init_params.py.

11.9.3.3 basis

`init_params.basis = tk.IntVar()`

Definition at line 184 of file init_params.py.

11.9.3.4 bg_button

`string init_params.bg_button = 'grey'`

note: Mac OS may not support changing button color

Parameters

bg_button	background color of the close button

Definition at line 35 of file init_params.py.

11.9.3.5 bg_color

```
string init_params.bg_color = 'white'
```

Parameters

bg_color	background color of the gui window

Definition at line 30 of file init_params.py.

11.9.3.6 bopt

```
init_params.bopt = tk.IntVar()
```

Definition at line 164 of file init_params.py.

11.9.3.7 close

```
init_params.close
```

Initial value:

```
= tk.Button(window,text='Close',command=window.quit,
                bg=bg_button,fg=fg_button)
```

Definition at line 212 of file init_params.py.

11.9.3.8 column

```
init_params.column
```

Definition at line 83 of file init_params.py.

11.9.3.9 columnspan

```
init_params.columnspan
```

Definition at line 83 of file init_params.py.

11.9.3.10 def_values

```
list init_params.def_values = [1,1,1.5,40,1000000,0,10,100,1,10,12345,1,7,0.01,1.0,1.0,5.0,20]
```

Parameters

def_values	Default parameter values carried over from latin_driver.f90

Definition at line 132 of file init_params.py.

11.9.3.11 dval

```
init_params.dval
```

Definition at line 150 of file init_params.py.

11.9.3.12 entry5

```
init_params.entry5 = bopt
```

Definition at line 171 of file init_params.py.

11.9.3.13 entry7

```
init_params.entry7 = basis
```

Definition at line 190 of file init_params.py.

11.9.3.14 fg_button

```
string init_params.fg_button = 'black'
```

Parameters

fg_color	foreground color of the close button

Definition at line 37 of file init_params.py.

11.9.3.15 fg_color

```
string init_params.fg_color = 'grey'
```

Parameters

fg_color	foreground color of the gui window

Definition at line 32 of file init_params.py.

11.9.3.16 file

```
init_params.file = open(filename,'w')
```

Definition at line 266 of file init_params.py.

11.9.3.17 filename

```
string init_params.filename = 'init_params.txt'
```

Definition at line 265 of file init_params.py.

11.9.3.18 floats

```
list init_params.floats = [2, 13, 14, 15, 16]
```

Parameters

floats	Indices of parameters that are floats

Definition at line 135 of file init_params.py.

11.9.3.19 font

```
string init_params.font = 'Times New Roman'
```

Parameters

font	Font of the GUI text

Definition at line 40 of file init_params.py.

11.9.3.20 fontsize

```
int init_params.fontsize = 18
```

Parameters

fontsize	Size of GUI font

Definition at line 42 of file init_params.py.

11.9.3.21 height

```
int init_params.height = 1000
```

Parameters

height	Height of the GUI window

Definition at line 47 of file init_params.py.

11.9.3.22 i

```
int init_params.i = 6
```

Definition at line 173 of file init_params.py.

11.9.3.23 indicatoron

```
init_params.indicatoron
```

Definition at line 166 of file init_params.py.

11.9.3.24 integers

```
list init_params.integers = [0, 1, 3, 4, 5, 6, 7,8, 9, 10, 11, 12, 17]
```

Parameters

integers	Indices of parameters that are iNtegers

Definition at line 137 of file init_params.py.

11.9.3.25 length

```
int init_params.length = 1000
```

Parameters

length	Length of the GUI window

Definition at line 45 of file init_params.py.

11.9.3.26 N_params

```
init_params.N_params = len(params)
```

Parameters

N_params	Number of parameters/variables

Definition at line 140 of file init_params.py.

11.9.3.27 p1

```
int init_params.p1 = 1
```

Checks to see if user input is proper and sets everything to either an integer or float.

Definition at line 222 of file init_params.py.

11.9.3.28 p2

```
int init_params.p2 = 0
```

Definition at line 256 of file init_params.py.

11.9.3.29 padx

```
init_params.padx
```

Definition at line 167 of file init_params.py.

11.9.3.30 pady

```
init_params.pady
```

Definition at line 156 of file init_params.py.

11.9.3.31 params

```
list init_params.params
```

Initial value:

```
= ['Number of electrons','Number of atoms','Bond length if there are two atoms', \
'Number of trials for latin hypercube search','Total number of steps for MCMC run', \
'Biased optimizer','Number of steps', \
'Type of orbitals', \
'Proton number', \
'Number of OMP Threads', \
'Random seed for the latin hypecube', \
'Number of linear terms in the single-electron wavefunction per atom', \
'Number of dofs in the Jastrow (interaction) term', \
'Lengthscale for the finite difference method', \
'Inverse lengthscale of nuclear-electron interaction', \
'Inverse lengthscale of electron-electron interaction', \
'Amount of distance away from atoms to plot', \
'Number of points along axis in each direction to plot']
```

Parameters

params	The names of the parameters needed, given in the order of entry.

Definition at line 116 of file init_params.py.

11.9.3.32 relx

```
init_params.relx
```

Definition at line 214 of file init_params.py.

11.9.3.33 rely

```
init_params.rely
```

Definition at line 214 of file init_params.py.

11.9.3.34 row

```
init_params.row
```

Definition at line 83 of file init_params.py.

11.9.3.35 sticky

```
init_params.sticky
```

Definition at line 83 of file init_params.py.

11.9.3.36 text

```
init_params.text
```

Definition at line 150 of file init_params.py.

11.9.3.37 text0

```
init_params.text0 = tk.Label(window,text=txt0,font=(font,fontsize),wraplength=length-5)
```

Initializing/Configuring the top row text.

Definition at line 81 of file init_params.py.

11.9.3.38 text1

```
init_params.text1 = tk.Label(window,text=txt1,font=(font,fontsize),wraplength=length-10)
```

Definition at line 143 of file init_params.py.

11.9.3.39 text2

```
init_params.text2 = tk.Label(window,text=txt2,font=(font,fontsize),wraplength=length-10)
```

Definition at line 154 of file init_params.py.

11.9.3.40 text2_1

```
init_params.text2_1 = tk.Label(window,text=txt2_1,font=(font,fontsize),wraplength=length-10)
```

Definition at line 160 of file init_params.py.

11.9.3.41 text3

```
init_params.text3 = tk.Label(window,text=txt3,font=(font,fontsize),wraplength=length-10)
```

Definition at line 180 of file init_params.py.

11.9.3.42 text4

```
init_params.text4 = tk.Label(window,text=txt4,font=(font,fontsize),wraplength=length-10)
```

Definition at line 200 of file init_params.py.

11.9.3.43 title

```
string init_params.title = 'User Input'
```

Parameters

title	GUI window title

Definition at line 27 of file init_params.py.

11.9.3.44 txt0

```
string init_params.txt0
```

Initial value:
```
= "The default simulation parameters are shown in the boxes in \
the right-hand column below. Please change any to your desired simulation \
parameters and then click 'Close' at the bottom."
```

Parameters

txt0	The text shown in the top row of the GUI

Definition at line 76 of file init_params.py.

11.9.3.45 txt1

```
string init_params.txt1 = "The user must provide the following inputs:  "
```

Definition at line 142 of file init_params.py.

11.9.3.46 txt2

```
string init_params.txt2 = "Biased Optimizer Options (not enabled by default):"
```

Definition at line 153 of file init_params.py.

11.9.3.47 txt2_1

```
string init_params.txt2_1 = "Enable the biased optimizer?"
```

Create a radio button for user to choose between two options.

Definition at line 159 of file init_params.py.

11.9.3.48 txt3

```
string init_params.txt3 = "Optional user inputs (default options are shown):"
```

Definition at line 179 of file init_params.py.

11.9.3.49 txt4

```
string init_params.txt4 = "Visualization options:"
```

Definition at line 199 of file init_params.py.

11.9.3.50 value

```
init_params.value
```

Definition at line 166 of file init_params.py.

11.9.3.51 variable

```
init_params.variable
```

Definition at line 166 of file init_params.py.

11.9.3.52 window

```
init_params.window = tk.Tk()
```

Definition at line 51 of file init_params.py.

11.10 log_rho_mod Module Reference

Driver for testng the biased optimization routines.

Functions/Subroutines

- real(dp) function log_rho (x, dof)

11.10.1 Detailed Description

Driver for testng the biased optimization routines.

11.10.2 Function/Subroutine Documentation

11.10.2.1 log_rho()

```
real(dp) function log_rho_mod::log_rho (
            real(dp), dimension(:), intent(in) x,
            real(dp), dimension(:), intent(in) dof )
```

Definition at line 11 of file Biased_Optim_example_driver.f90.

```
11     implicit none
12     real(dp), dimension(:), intent(in) :: x
13     real(dp), dimension(:), intent(in) :: dof
14     real(dp) :: retval
15 
16 
17     retval = 2*log(abs(wave_function(x,dof)))
18 
```

References basis_functions::wave_function.

Referenced by main().

Here is the caller graph for this function:

11.11 mcmc Module Reference

Functions and subroutines for implementing Markov chain Monte Carlo.

Data Types

- interface log_rho_interface

Functions/Subroutines

- subroutine mcmc_sample (samples, log_rho, x_0, n_steps, n_burned, thinning_interval, s, e_code, dof_↩ coefficients, density_dimension, average_accept, seed)

  The main routine for MCMC, generates (n_steps-n_burned)/thinning_interval samples from a distribution rho.
- subroutine mcmc_adapt (s_out, log_rho, x_0, n_steps, s_0, e_code, s_max, s_min, memory, adapt_interval, dof_coefficients, density_dimension, seed)

  Runs a version of mcmc_sample but every adapt_interval steps adjusts s, based on an exponetial average of the accpetance rate.

11.11.1 Detailed Description

Functions and subroutines for implementing Markov chain Monte Carlo.

11.11.2 Function/Subroutine Documentation

11.11.2.1 mcmc_adapt()

```
subroutine mcmc::mcmc_adapt (
            real(dp), intent(out) s_out,
            real(dp), dimension(:), intent(in) log_rho,
            real(dp), dimension(density_dimension), intent(in) x_0,
            integer, intent(in) n_steps,
            real(dp), intent(in) s_0,
            integer, intent(out) e_code,
            real(dp), intent(in) s_max,
            real(dp), intent(in) s_min,
            real(dp), intent(in) memory,
            integer, intent(in) adapt_interval,
            real(dp), dimension(:), intent(in) dof_coefficients,
            integer, intent(in) density_dimension,
            integer, dimension(:), intent(in), optional seed )
```

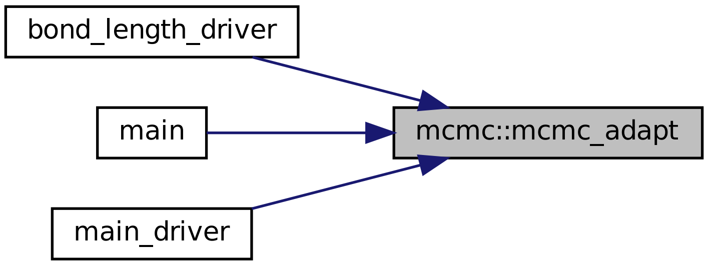
Runs a version of mcmc_sample but every adapt_interval steps adjusts s, based on an exponetial average of the accpetance rate.

Definition at line 194 of file MCMC.f90.

```
    implicit none
    real(dp), intent(out) :: s_out
    procedure(log_rho_interface) :: log_rho
    integer, intent(in) :: density_dimension
    integer, intent(in) :: n_steps
    integer, intent(in) :: adapt_interval
    integer, intent(in), dimension(:), optional :: seed
    real(dp), dimension(:), intent(in) :: dof_coefficients
    real(dp), intent(in) :: s_0
    real(dp), intent(in) :: s_max, s_min
    real(dp), intent(in) :: memory
    real(dp), dimension(density_dimension), intent(in) :: x_0
    integer, intent(out) :: e_code
    real(dp), dimension(density_dimension) :: x,x_prop
    real(dp) :: a,a_unnorm,norm_cnst,a_ave,rand_sample,log_rho_x,log_rho_x_prop,s, factor, a_ave_
    real(dp) :: a_min, a_max !keep in close range about 0.235
    integer :: loop_index, adapt_count

    a_min = 0.234_dp
    a_max = 0.236_dp
    x = x_0
    a_unnorm = 0.0_dp
    log_rho_x = log_rho(x_0,dof_coefficients)
    s=s_0
    norm_cnst=1.0_dp
    a_ave_=1.0_dp
    factor=sqrt(2.0_dp)

    !uses current random state if none is given
    if (present(seed)) then
        call random_seed(put=seed)
    end if

    do loop_index=1,n_steps
        x_prop = mcmc_propose(x,s,density_dimension)
        call mcmc_accept(a, x, x_prop, log_rho, s, log_rho_x, log_rho_x_prop,dof_coefficients)
        a = exp(a)
        call random_number(rand_sample)
        if (a>rand_sample) then
            x = x_prop
            log_rho_x = log_rho_x_prop !using datalog_rho_x_prop from last loop, saves calls to rho
            !computing average acceptance rate, will want to have way of outputing at some point
            a_unnorm = 1.0_dp + exp(-1.0_dp/memory)*a_unnorm
            a_ave_= (1.0_dp+(loop_index-1)*a_ave_)/loop_index
        else
            a_unnorm = exp(-1.0_dp/memory)*a_unnorm
            a_ave_= ((loop_index-1)*a_ave_)/loop_index
        end if
        norm_cnst = 1.0_dp + exp(-1.0_dp/memory)*norm_cnst

        adapt_count = adapt_count+1
        if (adapt_count .ge. adapt_interval) then
            if (a_ave_.le.0) then
                if (a_ave_<0) then
                    e_code=e_code+2**1
                    print*, 'ERROR: Acceptance rate negative at:', loop_index
                    EXIT
                end if
                print*, 'WARNING: Acceptance rate 0 at:', loop_index
            end if
            adapt_count=0
            a_ave = a_unnorm/norm_cnst
            if (a_ave<a_min) then
                s=s/factor
                if (s<s_min) then
                    s=s_min
                end if
            else if (a_ave>a_max) then
                s=s*factor
                if (s>s_max) then
                    s=s_max
                end if
            end if
        end if
    end do

    s_out=s
    !e_code 0 => no error
    e_code=0

```

Referenced by bond_length_driver(), main(), and main_driver().

Here is the caller graph for this function:



11.11.2.2 mcmc_sample()

```
subroutine mcmc::mcmc_sample (
             real(dp), dimension((n_steps-n_burned)/thinning_interval+1,density_dimension),
intent(out) samples,
             real(dp), dimension(:), intent(in) log_rho,
             real(dp), dimension(density_dimension), intent(in) x_0,
             integer, intent(in) n_steps,
             integer, intent(in) n_burned,
             integer, intent(in) thinning_interval,
             real(dp), intent(in) s,
             integer, intent(out) e_code,
             real(dp), dimension(:), intent(in) dof_coefficients,
             integer, intent(in) density_dimension,
             real(dp), intent(out), optional average_accept,
             integer, dimension(:), intent(in), optional seed )
```

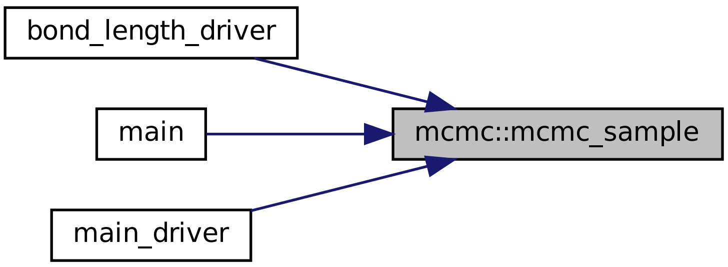
The main routine for MCMC, generates (n_steps-n_burned)/thinning_interval samples from a distribution rho.

Definition at line 111 of file MCMC.f90.

```
        implicit none
        procedure(log_rho_interface) :: log_rho
        integer, intent(in) :: density_dimension
        integer, intent(in) :: n_steps
        integer, intent(in) :: n_burned
        integer, intent(in) :: thinning_interval
        integer, intent(in), dimension(:), optional :: seed
        real(dp), dimension(:), intent(in) :: dof_coefficients
        real(dp), intent(in) :: s
        real(dp), dimension(density_dimension), intent(in) :: x_0
        integer, intent(out) :: e_code
        real(dp), intent(out), optional :: average_accept
        real(dp), dimension((n_steps-n_burned)/thinning_interval+1,density_dimension), intent(out) ::
    samples
        real(dp), dimension(density_dimension) :: x,x_prop
        real(dp) :: a,a_ave,rand_sample,log_rho_x,log_rho_x_prop
        integer :: loop_index,out_array_index,thinning_counter

        !e_code 0 => no error
        e_code=0

        x = x_0
        out_array_index = 0
        thinning_counter = 0
        samples(1,:) = x_0
        a_ave = 0.0_dp
        log_rho_x = log_rho(x_0,dof_coefficients)

        !uses current random state if none is given
        if (present(seed)) then
            call random_seed(put=seed)
        end if

        do loop_index=1,n_steps
            x_prop = mcmc_propose(x,s,density_dimension)
            call mcmc_accept(a, x, x_prop, log_rho, s,log_rho_x,log_rho_x_prop, dof_coefficients)
            a = exp(a)
            call random_number(rand_sample)
            if (a>rand_sample) then
                x = x_prop
                log_rho_x = log_rho_x_prop !using datalog_rho_x_prop from last loop, saves calls to rho
                !computing average acceptance rate, will want to have way of outputing at some point
                a_ave = (1.0_dp/loop_index)*(a_ave*(loop_index-1.0_dp)+1.0_dp)
            else
                a_ave = (1.0_dp/loop_index)*(a_ave*(loop_index-1))
            end if
            if (loop_index.ge.n_burned) then
                thinning_counter = thinning_counter+1
            end if
            if (thinning_counter .ge. thinning_interval) then
                out_array_index = out_array_index+1
                if (out_array_index>(n_steps-n_burned)/thinning_interval) then
                    e_code = e_code+2**2
                    print*, 'ERROR: Output larger than output array', loop_index
                    exit
                end if
                if (a.ne.a) then
                    e_code=e_code+1
                    print*, 'ERROR: NAN acceptance at:', loop_index
                    EXIT
                end if
                if (a_ave.le.0) then
                    if (a_ave<0) then
                        e_code=e_code+2**1
                        print*, 'ERROR: Acceptance rate negative at:', loop_index
                        EXIT
                    end if
                    print*, 'WARNING: Acceptance rate 0 at:', loop_index
                end if
                samples(out_array_index,:) = x
                thinning_counter=0
            end if
        end do


        if (present(average_accept)) then
            average_accept = a_ave
        end if

```

Referenced by bond_length_driver(), main(), and main_driver().

Here is the caller graph for this function:



11.12 param_search Module Reference

Functions/subroutines associated with building/initialisation the parameter search space and finding the best paramters from a given MCMC run.

Functions/Subroutines

- subroutine random_search_grid (trials, param_bounds, n_trials, seed_in)

  random_search_grid returns MxN grid of test points for M free parameters and N trials distributed
- subroutine latin_hypercube (trials, param_bounds, n_trials, seed_in)

  latin_hypercube returns MxN grid of test points for M free parameters and N trials on a latin hypercube for given bounds.
- subroutine find_best_params (trial_energies, trials)

  find_best_params returns the 1D array that contain the current best set of parameters found from a MCMC run.
- subroutine param_wall_check (param_bounds)

  param_wall_check evaluates if the best parameter set is close to the parameter bounds used to generate the test points.

Variables

- real(dp), dimension(:), allocatable, protected best_params
- integer, dimension(1), protected best_trial

11.12.1 Detailed Description

Functions/subroutines associated with building/initialisation the parameter search space and finding the best paramters from a given MCMC run.

11.12.2 Function/Subroutine Documentation

11.12.2.1 find_best_params()

```
subroutine param_search::find_best_params (
            real(dp), dimension(:), intent(in), allocatable trial_energies,
            real(dp), dimension(:, :), intent(in), allocatable trials )
```

find_best_params returns the 1D array that contain the current best set of parameters found from a MCMC run.

the output is stored in the global variable best_params.

Parameters

trial_energies	is a 1D array of the energies found for each parameter set.
trials	is a 2D array containing the parameter set for each trial used in MCMC.

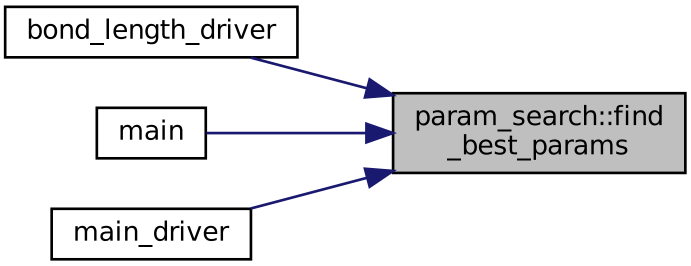
Definition at line 107 of file param_search.f90.

```
    real(dp), dimension(:, :), allocatable, intent(in) :: trials
    real(dp), dimension(:), allocatable, intent(in) :: trial_energies
    integer, dimension(1) :: arr_shape
    arr_shape = shape(trials(1,:))
    if(allocated(best_params)) deallocate(best_params)
    allocate(best_params(arr_shape(1)))
    best_trial = minloc(trial_energies) !index corresponding to the most negative, "best", energy.
    best_params = trials(best_trial(1),:)
```

References best_params, and best_trial.

Referenced by bond_length_driver(), main(), and main_driver().

Here is the caller graph for this function:



11.12.2.2 latin_hypercube()

```
subroutine param_search::latin_hypercube (
            real(dp), dimension(:, :), intent(inout) trials,
            real(dp), dimension(:,:), intent(in) param_bounds,
            integer, intent(in) n_trials,
            integer, intent(in) seed_in )
```

latin_hypercube returns MxN grid of test points for M free parameters and N trials on a latin hypercube for given bounds.

Parameters

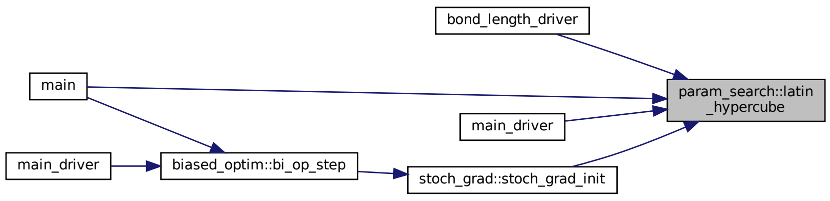
trials	is a 2D array containing the parameter set for each trial used in MCMC.
param_bounds	is a 2D array containing the upper and lower bounds for each parameter in the wavefunciton.
n_trials	is the number of trials to be tested in MCMC. This defines the binning of the Latin hypercube.
seed_in	is an integer that defines the seed being used to generate the latin Latin hypercube.

Definition at line 55 of file param_search.f90.

```
    real(dp), dimension(:,:), intent(in) :: param_bounds !length of this will be the "N"
    real(dp), dimension(:, :),intent(inout) :: trials ! (x,y) x is each trial, y is each param
    integer, intent(in) :: n_trials, seed_in !n_trials is the "M" and is specified as an input, seed_in
    is an integer used to set the seed.
    real(dp), dimension(:),allocatable ::  zero_to_one
    real(dp) :: rand_num, temp_val
    integer, dimension(2) :: arr_shape
    integer :: i,j, k, h, n
    integer,allocatable :: seed(:)
    !setting a certain seed is done as shown below.
    call random_seed(size=n)
    allocate(seed(n))
    seed = seed_in    !set seed value to that read in from input.
    call random_seed(put=seed) !actually set the random seed for repeatability


    ! arr_shape(1) contains the number of variables, param_bounds(x,y) where x is for each param and y
    for limits.
    arr_shape = shape(param_bounds)

    allocate(zero_to_one(n_trials))
    do i = 1,n_trials
      zero_to_one(i) = (real(i, dp)-1.0_dp)/(n_trials-1.0_dp) !extra bit needed to make sure we are
    binning right in the param search space!
    end do

    ! initialize latin cube in order.
    do i = 1,arr_shape(1)
      trials(:, i) = zero_to_one*(param_bounds(i,1) - param_bounds(i,2)) + param_bounds(i,2)
    end do

    !scramble the array for each param.
    do h = 1, 2 !sufficient scrambling.
      do i = 1,arr_shape(1) !loop for each parameter.
        do j = 1, n_trials !scramble over this loop
          call random_number(rand_num)
          k = 1 + floor((n_trials+1-1)*rand_num) !this is how we are randomly selecting an index to swap.
          temp_val=trials(k,i) !store temp value to swap
          trials(k,i) = trials(j,i) !swap points in the grid
          trials(j,i) = temp_val !swap back the temp value. simple as.
        end do
      end do
    end do

    deallocate(zero_to_one,seed)
```

Referenced by bond_length_driver(), main(), main_driver(), and stoch_grad::stoch_grad_init().

Here is the caller graph for this function:



11.12.2.3 param_wall_check()

```
subroutine param_search::param_wall_check (
              real(dp), dimension(:,:), intent(in), allocatable param_bounds )
```

param_wall_check evaluates if the best parameter set is close to the parameter bounds used to generate the test points.

The tolerence is set at 5%.

Parameters

param_bounds	is a 2D array containing the upper and lower bounds for each parameter in the wavefunciton.

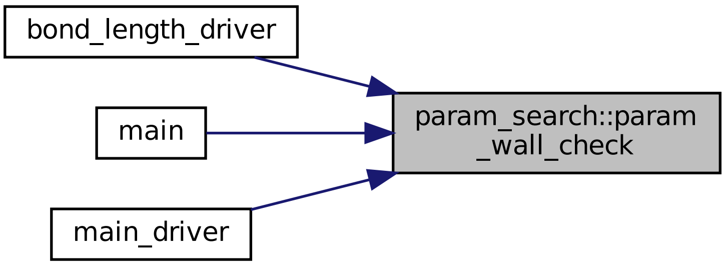
Definition at line 121 of file param_search.f90.

```
    real(dp), dimension(:,:), allocatable ,intent(in) :: param_bounds
    real(dp), parameter :: tol = 0.05_dp !percentage tol as determined by the min and max bounds.
    real(dp) :: bound_gap
    integer :: i

    do i = 1, size(best_params)
      bound_gap = abs(param_bounds(i,1) - param_bounds(i,2))
      if ( abs(best_params(i) - param_bounds(i,1))/bound_gap < tol &
      .or. abs(best_params(i) - param_bounds(i,2))/bound_gap < tol ) then
        print*, "parameter ", i,  " is close to wall"
        !for later: might want to instead store i in and array of potential problem parameters.
        !As oppsed to just printing out the problem.
      end if
    end do
```

References best_params.

Referenced by bond_length_driver(), main(), and main_driver().

Here is the caller graph for this function:



11.12.2.4 random_search_grid()

```
subroutine param_search::random_search_grid (
            real(dp), dimension(:, :), intent(inout) trials,
            real(dp), dimension(:,:), intent(in), allocatable param_bounds,
            integer, intent(in) n_trials,
            integer, intent(in) seed_in )
```

random_search_grid returns MxN grid of test points for M free parameters and N trials distributed

Parameters

trials	is a 2D array containing the parameter set for each trial used in MCMC.
param_bounds	is a 2D array containing the upper and lower bounds for each parameter in the wavefunciton.
n_trials	is the number of trials to be tested in MCMC. This defines the binning of the Latin hypercube.
seed_in	is an integer that defines the seed being used to generate the latin Latin hypercube.

Definition at line 19 of file param_search.f90.

```
    real(dp), dimension(:,:), allocatable ,intent(in) :: param_bounds !length of this will be the "N"
    real(dp), dimension(:, :),intent(inout) :: trials ! (x,y) x is each trial, y is each param
    integer, intent(in) :: n_trials, seed_in!this is the "M"
    real(dp), dimension(:),allocatable :: rand_num
    integer, dimension(2) :: arr_shape
    integer :: i, n
    integer,allocatable :: seed(:)
    !initialising a certain seed is done as shown below.
    call random_seed(size=n)
    allocate(seed(n))
    seed = seed_in   !set seed to that read in from input.
    call random_seed(put=seed)


    ! arr_shape(1) contains the number of variables, param_bounds(x,y) where x is for each param and y
    for limits.
    arr_shape = shape(param_bounds)

    allocate(rand_num(n_trials))

    do i = 1,arr_shape(1)
    call random_number(rand_num)
    trials(:, i) = rand_num*(param_bounds(i,1) - param_bounds(i,2)) + param_bounds(i,2)
    end do

```

11.12.3 Variable Documentation

11.12.3.1 best_params

`real(dp), dimension(:), allocatable, protected param_search::best_params`

Definition at line 10 of file param_search.f90.

```
real(dp), dimension(:), allocatable, protected :: best_params
```

Referenced by find_best_params(), main(), main_driver(), and param_wall_check().

11.12.3.2 best_trial

`integer, dimension(1), protected param_search::best_trial`

Definition at line 11 of file param_search.f90.

```
integer, dimension(1), protected :: best_trial
```

Referenced by bond_length_driver(), find_best_params(), main(), and main_driver().

11.13 plotting Namespace Reference

Main plotting script which outputs the contour plots for the wavefunction or electron probability density.

Variables

- coords = np.loadtxt('xyz.txt')

  Uploading the xy plane coordinates from the xyz.txt file generated from the xy_grid subroutine in netcdf_file.f90.
- data = nc.Dataset('results.nc4', mode='r', format='NETCDF4')

  Uploading the main results NetCDF file named results.nc4 generated from the result_netcdf routine.
- num_ele = data.variables['Num_of_Electrons'][:]

  Prints a statement if it has been uploaded successfully.
- num_nuc = data.variables['Num_of_Nuclei'][:]
- x = coords[:,0]

  Extracts the x and y coordinates from the xyz.txt.
- y = coords[:,1]
- wavefunction = data.variables['Electron_Density'][:]

  This does the contour plot for the 1 electron case.
- dof = data.variables['Optimal_DOF'][:]
- ele_den = data.variables['Electron_Density'][:]

  Creates the contour plot of the wavefunction squared.

11.13.1 Detailed Description

Main plotting script which outputs the contour plots for the wavefunction or electron probability density.

11.13.2 Variable Documentation

11.13.2.1 coords

```
plotting.coords = np.loadtxt('xyz.txt')
```

Uploading the xy plane coordinates from the xyz.txt file generated from the xy_grid subroutine in netcdf_file.f90.

Definition at line 11 of file plotting.py.

11.13.2.2 data

```
plotting.data = nc.Dataset('results.nc4', mode='r', format='NETCDF4')
```

Uploading the main results NetCDF file named results.nc4 generated from the result_netcdf routine.

Parameters

data	this contains all the information contained in the NetCDF file

Definition at line 15 of file plotting.py.

11.13.2.3 dof

```
plotting.dof = data.variables['Optimal_DOF'][:]
```

Parameters

dof	contains the degrees of freedom data
dof	the degrees of freedom

Definition at line 46 of file plotting.py.

11.13.2.4 ele_den

```
plotting.ele_den = data.variables['Electron_Density'][:]
```

Creates the contour plot of the wavefunction squared.

This does the contour plot for the 2 electron case. It returns a contour plot of the electron probability density in the xy plane

Parameters

ele_den	the electron probability density

Definition at line 62 of file plotting.py.

11.13.2.5 num_ele

```
plotting.num_ele = data.variables['Num_of_Electrons'][:]
```

Prints a statement if it has been uploaded successfully.

Extracting the number of electrons and nuclei variables from the result file

Parameters

num_ele	is the number of electrons used

Definition at line 25 of file plotting.py.

11.13.2.6 num_nuc

```
plotting.num_nuc = data.variables['Num_of_Nuclei'][:]
```

Parameters

num_nuc	is the number of nuclei

Definition at line 28 of file plotting.py.

11.13.2.7 wavefunction

```
plotting.wavefunction = data.variables['Electron_Density'][:]
```

This does the contour plot for the 1 electron case.

It returns a contour plot of the wavefunction squared in the xy plane

Extracts the wavefunction and degrees of freedom parameters from the input NetCDF file

Parameters

wavefunction	contains the wavefunction data

Definition at line 44 of file plotting.py.

11.13.2.8 x

```
plotting.x = coords[:,0]
```

Extracts the x and y coordinates from the xyz.txt.

Parameters

x	is the x coordinates
y	is the y coordinates

Definition at line 35 of file plotting.py.

11.13.2.9 y

```
plotting.y = coords[:,1]
```

Definition at line 36 of file plotting.py.

11.14 priors Module Reference

Functions for obtaining the prior mean and its derivative.

Functions/Subroutines

- real(dp) function prior_mean (x)

  Gives the prior mean.
- real(dp) function prior_mean_dx (x, dim)

  Derivative of the prior mean.

11.14.1 Detailed Description

Functions for obtaining the prior mean and its derivative.

11.14.2 Function/Subroutine Documentation

11.14.2.1 prior_mean()

```
real(dp) function priors::prior_mean (
              real(dp), dimension(:), intent(in) x )
```

Gives the prior mean.

Definition at line 9 of file priors.f90.

```
          implicit none
           real(dp), dimension(:), intent(in) :: x !param space point
           real(dp) :: out

           out = 1.0_dp

```

Referenced by main().

Here is the caller graph for this function:

main
priors::prior_mean

11.14.2.2 prior_mean_dx()

```
real(dp) function priors::prior_mean_dx (
              real(dp), dimension(:), intent(in) x,
              integer, intent(in) dim )
```

Derivative of the prior mean.

Definition at line 19 of file priors.f90.

```
           implicit none
           integer, intent(in) :: dim
           real(dp), dimension(:), intent(in) :: x !param space point
           real(dp) :: out

           out = 0.0_dp

```

Referenced by main().

Here is the caller graph for this function:

main
priors::prior_mean_dx

11.15 read_data Module Reference

Subroutines to read files and transfer user inputs into fortran.

Functions/Subroutines

- subroutine readtxt (filename, n_electrons_in, n_atoms_in, bond_length, n_trials, n_MCMC_steps, n_omp_↵ threads, use_biased_optimiser, n_optimiser_steps, basis_type_in, proton_number_int_in, search_seed, n_↵ basis_functions_per_atom_in, n_Jastrow_in, fd_length_in, Jastrow_b_length_in, Jastrow_d_length_in, plot↵ _distance, plot_points)

    Read in data from a text file.

11.15.1 Detailed Description

Subroutines to read files and transfer user inputs into fortran.

11.15.2 Function/Subroutine Documentation

11.15.2.1 readtxt()

```
subroutine read_data::readtxt (
             character(len=20) filename,
             integer n_electrons_in,
             integer n_atoms_in,
             real(dp) bond_length,
             integer n_trials,
             integer n_MCMC_steps,
             integer n_omp_threads,
             logical use_biased_optimiser,
             integer n_optimiser_steps,
             integer basis_type_in,
             integer proton_number_int_in,
             integer search_seed,
             integer n_basis_functions_per_atom_in,
             integer n_Jastrow_in,
             real(dp) fd_length_in,
             real(dp) Jastrow_b_length_in,
             real(dp) Jastrow_d_length_in,
             real(dp) plot_distance,
             integer plot_points )
```

Read in data from a text file.

This module reads txt, csv files line by line. The implementation is basic but will read each line properly as it was written specifically for the parameters needed from the user.

Parameters

filename	Filename of input text file
n_electrons_in	Number of electrons
n_atoms_in	Number of atoms
bond_length	Bond length for 2 atoms
n_trials	number of trials in the hypercube search (for each step of the biased optimiser)
n_mcmc_steps	Total number of steps for each MCMC run
n_omp_threads	Number of OMP Threads
use_biased_optimiser	Flag for use of biased optimiser: true uses biased optimiser, false just uses hypercube search
n_optimiser_steps	Number of steps in the optimser
n_basis_functions_per_atom←_in	Number of linear terms in the single-electron wavefunction per atom
search_seed	Seed for the latin hypercube
n_jastrow_in	Number of dofs in the Jastrow (interaction) term
fd_length_in	Lengthscale of the finite difference code
jastrow_b_length_in	Inverse lengthscale of nuclear-electron interaction
jastrow_d_length_in	Inverse lengthscale of electron-electron interaction
proton_number_int_in	Proton number of atoms
basis_type_in	integer code for type of basis. Codes defined in shared_constants.f90
plot_distance	Amount of distance away from atoms to plot
plot_points	Number of points along each axis to plot

Integer value to determine if biased optimiser should be used

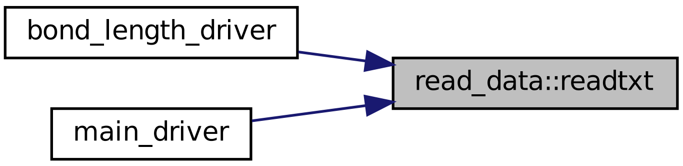
Definition at line 21 of file read_data.f90.

```

        ! User Inputs required - the user must define all of these
        ! Will obtain these from either txt file or a gui that generates the txt file

        CHARACTER(LEN=20) :: filename
        integer :: n_electrons_in
        integer :: n_atoms_in
        real(dp) :: bond_length
        integer :: n_trials
        integer :: n_MCMC_steps
        integer :: n_omp_threads
        logical :: use_biased_optimiser
        integer :: n_optimiser_steps
        ! Optional user Inputs - these may be defined by the user, but have good default values below
        integer :: n_basis_functions_per_atom_in
        integer :: search_seed
        integer :: n_Jastrow_in
```

```
      real(dp) :: fd_length_in
      real(dp) :: Jastrow_b_length_in
      real(dp) :: Jastrow_d_length_in
      integer :: proton_number_int_in
      integer :: basis_type_in

      real(dp) :: plot_distance
      integer :: plot_points

      integer :: bias_opt_0_1





     OPEN(unit = 7, file = filename)

     ! Read in lines containing only one value
     READ(7,*) n_electrons_in
     READ(7,*) n_atoms_in
     READ(7,*) bond_length
     READ(7,*) n_trials
     READ(7,*) n_mcmc_steps
     READ(7,*) bias_opt_0_1
     READ(7,*) n_optimiser_steps
     READ(7,*) basis_type_in
     READ(7,*) proton_number_int_in
     READ(7,*) n_omp_threads
     READ(7,*) search_seed
     READ(7,*) n_basis_functions_per_atom_in
     READ(7,*) n_jastrow_in
     READ(7,*) fd_length_in
     READ(7,*) jastrow_b_length_in
     READ(7,*) jastrow_d_length_in
     READ(7,*) plot_distance
     READ(7,*) plot_points

     CLOSE(7)

     if (bias_opt_0_1 == 1) then
       use_biased_optimiser = .true.
     else if ( bias_opt_0_1 == 0 ) then
        use_biased_optimiser = .false.
      end if


    ! Print to see if data was properly read into fortran
      ! PRINT *, n_electrons_in
      ! PRINT *, n_atoms_in
      ! PRINT *, bond_length
      ! PRINT *, n_trials
      ! PRINT *, n_MCMC_steps
      ! PRINT *, n_basis_functions_per_atom_in
      ! PRINT *, search_seed
      ! PRINT *, n_Jastrow_in
      ! PRINT *, fd_length_in
      ! PRINT *, Jastrow_b_length_in
      ! PRINT *, Jastrow_d_length_in

```

Referenced by bond_length_driver(), and main_driver().

Here is the caller graph for this function:



11.16 shared_constants Module Reference

Definitions of shared constants used throughout the software.

Variables

- integer, parameter dp = real64
- real(dp), parameter pi = 3.141592653589793238_dp
- real(dp), parameter electronvolt = 27.211386245988_dp
- integer, parameter slater_1s_code = 100
- integer, parameter sto_3g_code = 200

11.16.1 Detailed Description

Definitions of shared constants used throughout the software.

11.16.2 Variable Documentation

11.16.2.1 dp

```
integer, parameter shared_constants::dp = real64
```

Definition at line 6 of file constants.f90.

```
6   integer, parameter :: dp = real64
```

Referenced by bond_length_driver(), main(), and main_driver().

11.16.2.2 electronvolt

```
real(dp), parameter shared_constants::electronvolt = 27.211386245988_dp
```

Definition at line 8 of file constants.f90.

```
8   real(dp),parameter :: electronvolt = 27.211386245988_dp
```

Referenced by main(), and main_driver().

11.16.2.3 pi

```
real(dp), parameter shared_constants::pi = 3.141592653589793238_dp
```

Definition at line 7 of file constants.f90.

```
7   real(dp),parameter :: pi = 3.141592653589793238_dp
```

Referenced by component_functions::centered_gaussian(), component_functions::centered_slater_1s(), stoch_↩
grad::grad_accent(), and mcmc::log_rho_interface::log_rho_interface().

11.16.2.4 slater_1s_code

```
integer, parameter shared_constants::slater_1s_code = 100
```

Definition at line 11 of file constants.f90.

```
11   integer, parameter :: slater_1s_code = 100
```

Referenced by bond_length_driver(), basis_functions::initialise_basis(), and main_driver().

11.16.2.5 sto_3g_code

```
integer, parameter shared_constants::sto_3g_code = 200
```

Definition at line 12 of file constants.f90.

```
12   integer, parameter :: sto_3g_code = 200
```

Referenced by basis_functions::initialise_basis().

11.17 stoch_grad Module Reference

Optimization modules.

Functions/Subroutines

- subroutine stoch_grad_init (N, best_from_last_run_in, seed)

  Initalises module variables and sets up starts points for the restarts of the gradient assent.
- subroutine stoch_grad_exit ()

  deallocates state variables
- subroutine grad_accent (X_0, sequence_length, no_samples, gamma, out)

  Performs the gradient assent for sequence length X_0 is the intial point to start the assent at sequence_length is how long the sequence to average over is No_samlples is the number of samples taken in the gradient estimator, gamma is parameter to tune for how much to move the next point long the estimated gradient out is the output and used to compute the sum on the go.
- real(dp) function, dimension(x_shape(1), x_shape(2)) g_t (X, no_samples, X_shape)

  Return an estimate for the gradient which is an average over no_samples.
- real(dp) function, dimension(:,:), allocatable nearest_point (bounds, X)

  Returns a new point back to the search space if it leaves.

Variables

- integer n_restarts
- real(dp), dimension(:,:,:), allocatable n_points

11.17.1 Detailed Description

Optimization modules.

11.17.2 Function/Subroutine Documentation

11.17.2.1 g_t()

```
real(dp) function, dimension(x_shape(1),x_shape(2)) stoch_grad::g_t (
            real(dp), dimension(:,:), intent(in) X,
            integer no_samples,
            integer, dimension(2), intent(in) X_shape )
```

Return an estimate for the gradient which is an average over no_samples.

Parameters

X	initial input
no_samples	how many samples you want to use from the gradient estimator
X_shape	defines the shape of the output

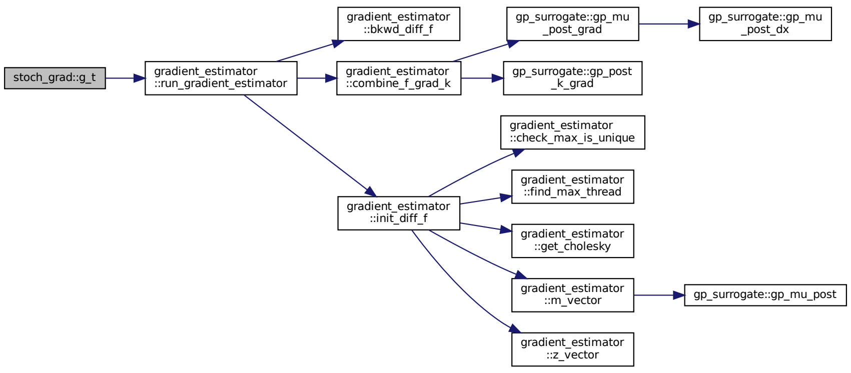
Definition at line 116 of file stoch_grad.f90.

```
116
117     implicit none
118     INTEGER:: no_samples
119     real(dp), dimension(:,:), intent(in) :: X
120     integer, dimension(2), intent(in) :: X_shape
121     real(dp), DIMENSION(X_shape(1),X_shape(2)) :: G_t
122     INTEGER :: i
123
124     g_t=x
125
126     do i = 1,no_samples
127
128         g_t = g_t + run_gradient_estimator(g_t,gaussian(size(x,1)),n_dof,best_from_last_run)
129
130     end do
131
132     g_t = g_t/no_samples
133
```

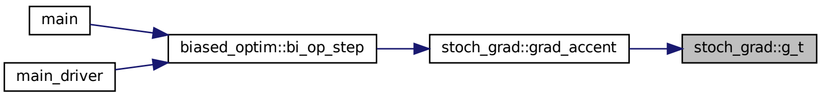
References gradient_estimator::run_gradient_estimator().

Referenced by grad_accent().

Here is the call graph for this function:


stoch_grad::g_t
gradient_estimator ::run_gradient_estimator
gradient_estimator ::bkwd_diff_f
gradient_estimator ::combine_f_grad_k
gp_surrogate::gp_mu _post_grad
gp_surrogate::gp_mu _post_dx
gp_surrogate::gp_post _k_grad
gradient_estimator ::init_diff_f
gradient_estimator ::check_max_is_unique
gradient_estimator ::find_max_thread
gradient_estimator ::get_cholesky
gradient_estimator ::m_vector
gp_surrogate::gp_mu_post
gradient_estimator ::z_vector

Here is the caller graph for this function:



11.17.2.2 grad_accent()

```
subroutine stoch_grad::grad_accent (
            real(dp), dimension(:,:), intent(in) X_0,
            integer sequence_length,
            integer no_samples,
            real(dp) gamma,
            real(dp), dimension(:,:), intent(out) out )
```

Performs the gradient assent for sequence length X_0 is the intial point to start the assent at sequence_length is how long the sequence to average over is No_samlples is the number of samples taken in the gradient estimator, gamma is parameter to tune for how much to move the next point long the estimated gradient out is the output and used to compute the sum on the go.

as result is taken as average, computing on the go uses less memory

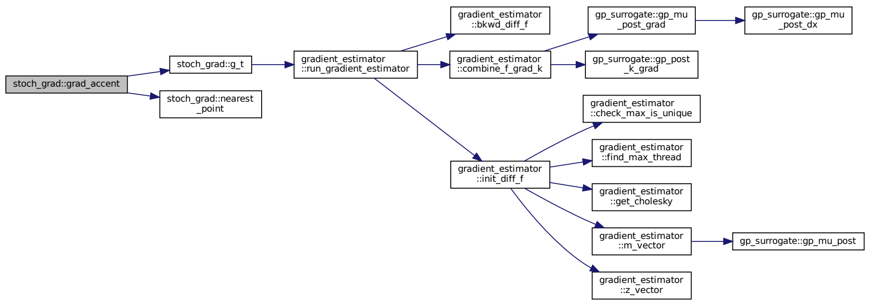
Definition at line 58 of file stoch_grad.f90.

```

     implicit none
     real(dp), dimension(:,:),intent(in) :: X_0
     real(dp), dimension(:,:), intent(out) :: out
     real(dp), dimension(:,:), allocatable :: X_t,X_t_prev
     real(dp) :: gamma
     INTEGER::i,sequence_length,no_samples

     ALLOCATE(x_t(size(x_0,1),size(x_0,2)))
     ALLOCATE(x_t_prev(size(x_0,1),size(x_0,2)))


     x_t_prev=x_0
     out = x_t_prev
     do i = 1,sequence_length

      x_t= nearest_point(dof_bounds,(x_t_prev + gamma*g_t(x_t_prev,no_samples,shape(x_t))))
      out = out + x_t
      x_t_prev = x_t

     end do

     out = out/real(sequence_length+1)

```

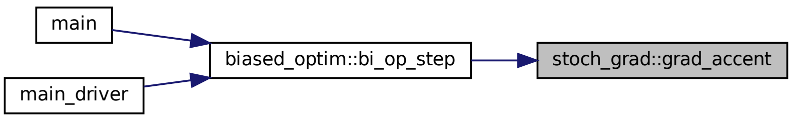
References basis_functions::dof_bounds, g_t(), nearest_point(), and shared_constants::pi.

Referenced by biased_optim::bi_op_step().

Here is the call graph for this function:


stoch_grad::grad_accent
stoch_grad::g_t
stoch_grad::nearest_point
gradient_estimator::run_gradient_estimator
gradient_estimator::bkwd_diff_f
gradient_estimator::combine_f_grad_k
gradient_estimator::init_diff_f
gp_surrogate::gp_mu_post_grad
gp_surrogate::gp_mu_post_dx
gp_surrogate::gp_post_k_grad
gradient_estimator::check_max_is_unique
gradient_estimator::find_max_thread
gradient_estimator::get_cholesky
gradient_estimator::m_vector
gp_surrogate::gp_mu_post
gradient_estimator::z_vector

Here is the caller graph for this function:



11.17.2.3 nearest_point()

```
real(dp) function, dimension(:,:), allocatable stoch_grad::nearest_point (
              real(dp), dimension(:,:), intent(in) bounds,
              real(dp), dimension(:,:), intent(in) X )
```

Returns a new point back to the search space if it leaves.

Parameters

X	new suggested point
bounds	the degree of freedom bounds defined on input

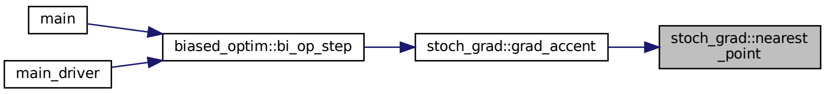
Definition at line 140 of file stoch_grad.f90.

```
140
141     implicit none
142     real(dp), dimension(:,:), intent(in) :: bounds
143     real(dp), dimension(:,:),INTENT(IN):: X
144     real(dp), dimension(:,:),ALLOCATABLE:: X_Out
145     INTEGER :: i,j
146
147     ALLOCATE(x_out(size(x,1),size(x,2)))
148
149     do i = 1,size(x,1)
150
151         do j = 1 ,size(x,2)
152
153             if (x(i,j)<bounds(j,1)) then
```

```
                x_out(i,j)=bounds(j,1)

            else if (x(i,j)>bounds(j,2)) then
                x_out(i,j)=bounds(j,2)
            else
                x_out(i,j) = x(i,j)
            end if

        end do



    end do
```

Referenced by grad_accent().

Here is the caller graph for this function:



11.17.2.4 stoch_grad_exit()

```
subroutine stoch_grad::stoch_grad_exit
```

deallocates state variables

Definition at line 44 of file stoch_grad.f90.

```
    implicit none

    deALLOCATE(n_points)

```

References n_points.

Referenced by main().

Here is the caller graph for this function:

main
stoch_grad::stoch_grad_exit

11.17.2.5 stoch_grad_init()

```
subroutine stoch_grad::stoch_grad_init (
              integer, intent(in) N,
              real(dp), intent(in) best_from_last_run_in,
              integer, dimension(:), intent(in) seed )
```

Initalises module variables and sets up starts points for the restarts of the gradient assent.

Parameters

N	sets number of restarts
best_from_last_run↩ _in	gives the best energy from the last set of points
seed	random seed in

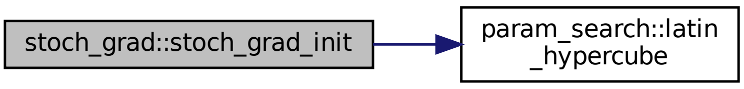
Definition at line 23 of file stoch_grad.f90.

```
    implicit none

    INTEGER,INTENT(IN) :: N
    real(dp), intent(in) :: best_from_last_run_in
    integer, dimension(:), INTENT(IN):: seed
    integer :: i

    n_dof = size(dof_bounds,1)
    n_restarts = n
    ALLOCATE(n_points(n_restarts,n_restarts,n_dof))

    best_from_last_run=best_from_last_run_in

    do i=1,n_restarts
        call latin_hypercube(n_points(i,:,:),dof_bounds, n_restarts ,seed(1))
    end do

```

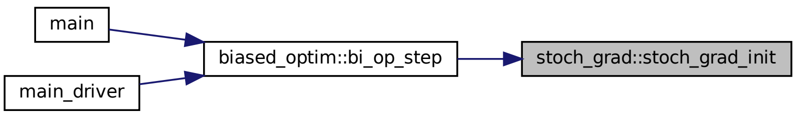
References basis_functions::dof_bounds, param_search::latin_hypercube(), n_points, and n_restarts.

Referenced by biased_optim::bi_op_step().

Here is the call graph for this function:



Here is the caller graph for this function:



11.17.3 Variable Documentation

11.17.3.1 n_points

```
real(dp), dimension(:,:,:), allocatable stoch_grad::n_points
```

Definition at line 13 of file stoch_grad.f90.

```
13     real(dp), DIMENSION(:,:,:), ALLOCATABLE :: N_points
```

Referenced by biased_optim::bi_op_step(), stoch_grad_exit(), and stoch_grad_init().

11.17.3.2 n_restarts

```
integer stoch_grad::n_restarts
```

Definition at line 11 of file stoch_grad.f90.

Referenced by stoch_grad_init().

11.18 write_file Module Reference

Contains all the subroutines related to writing results to file.

Functions/Subroutines

- subroutine result_netcdf (dof, ele, num_ele, num_nuc)

  Main results writing to NetCDF file.
- subroutine energies_netcdf (energies, bondlen)

  Output results for bond lengths and corresponding energy used for plotting.
- subroutine xy_grid (points, box_size)

  Writes the xyz coordinates used in the calculation of either the wavefunction or electron density to a text file names xyz.txt.
- subroutine write_restart_file (gp_n_data, gp_n_dof, n_cycles, no_samples, current_best_E, constant_↩ mean_value, gamma, kernel_var, kernal_inv_length, E_data, data_mean, param_pres, param_cov, param↩ _data)

  Creates a NetCDF file which contains information needed for the restart of the Bi_Op_step function.
- subroutine handle_err (ierr)

  Handles the NetCDF errors and print statements.

11.18.1 Detailed Description

Contains all the subroutines related to writing results to file.

This module contains all the subroutines needed for all the output result files which will be used for plotting and restarting.

11.18.2 Function/Subroutine Documentation

11.18.2.1 energies_netcdf()

```
subroutine write_file::energies_netcdf (
            real(dp), dimension(:), intent(in) energies,
            real(dp), dimension(:), intent(in) bondlen )
```

Output results for bond lengths and corresponding energy used for plotting.

Will only be used for the 2 nuclei case.

Parameters

filename	outputs a NetCDF file named bond_length.nc4

Dimension names

ID's

Inputting the data

Writing data to file

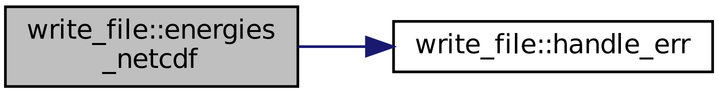
Closing the file

Definition at line 115 of file netcdf_file.f90.

```

    INTEGER, PARAMETER :: dp=kind(1.0d0)
    ! Inputting variables
    CHARACTER(LEN=100) :: filename = 'bond_length.nc4'
    REAL(dp), DIMENSION(:), INTENT(IN) :: energies, bondlen

    CHARACTER(LEN=100), DIMENSION(1) :: dim_e = (/"Energy"/)
    CHARACTER(LEN=100), DIMENSION(1) :: dim_b = (/"Bond_Lengths"/)

    ! File IDs
    INTEGER :: ierr, file_id
    ! Variable ID's
    INTEGER :: var_e, var_b
    ! Dimension ID's for the arrays
    INTEGER, DIMENSION(1) :: did_e, did_b
    ! Implicit types for the array sizes
    INTEGER, DIMENSION(1) :: size_e, size_b

    ! Create the file
    ierr = nf90_create(filename, nf90_clobber, file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Variable sizes
    size_e = shape(energies)
    size_b = shape(bondlen)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Energies
    ierr = nf90_def_dim(file_id, dim_e(1), size_e(1), did_e(1))
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    ierr = nf90_def_var(file_id, "Energy", nf90_double, did_e, var_e)

    ! Bond length
    ierr = nf90_def_dim(file_id, dim_b(1), size_b(1), did_b(1))
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    ierr = nf90_def_var(file_id, "Bond_length", nf90_double, did_b, var_b)

      ! Metadata
    ierr = nf90_enddef(file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Energy
    ierr = nf90_put_var(file_id, var_e, energies)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_b, bondlen)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_close(file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    !Finishing statement if writing file is successful.
    print *, "Success in writing the NETCDF file: ", filename

```

References handle_err().

Here is the call graph for this function:



11.18.2.2 handle_err()

```
subroutine write_file::handle_err (
            integer, intent(in) ierr )
```

Handles the NetCDF errors and print statements.

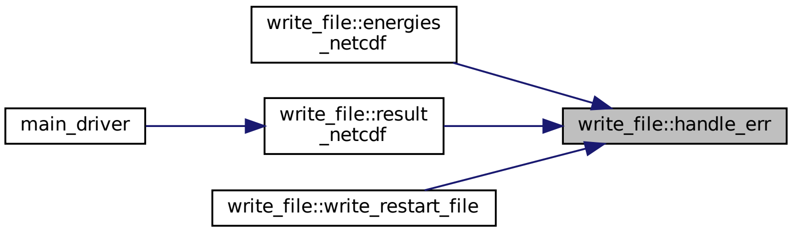
This subroutine is only used and called within this module.

Definition at line 436 of file netcdf_file.f90.

```

    INTEGER, INTENT(IN) :: ierr

    IF (ierr /= nf90_noerr) THEN
      print *, trim(nf90_strerror(ierr))
      ! Stops the code if an error is found
      stop "Stopped"
    END IF

```

Referenced by energies_netcdf(), result_netcdf(), and write_restart_file().

Here is the caller graph for this function:


write_file::energies_netcdf
main_driver
write_file::result_netcdf
write_file::handle_err
write_file::write_restart_file

11.18.2.3 result_netcdf()

```
subroutine write_file::result_netcdf (
            real(dp), dimension(:), intent(in) dof,
            real(dp), dimension(:), intent(in) ele,
            integer, intent(in) num_ele,
            integer, intent(in) num_nuc )
```

Main results writing to NetCDF file.

The result_netcdf subroutine outputs the main NetCDF result file containing the optimal degrees of freedom, either the electron density or wavefunction and the number of electrons and nuclei used. Saves the file as results.nc4 which is used Has error statements printed out at each section in order to aid with debugging.

Parameters

in	*dof*	dof the optimal degrees of freedom
in	*ele*	ele an rank 1 array containing either the wavefunction or electron density results evaluated on a grid op coordinates (for 1 or 2 electron system respectively).
in	*num_ele*	num_ele the number of electrons used in the calculation
	num_nuc	the number of nuclei used in the calculation
in	*num_nuc*	num_ele the number of electrons used in the calculation
	num_nuc	the number of nuclei used in the calculation

Dimension names for the arrays.

ID's

Inputting the data

Writing the data to file

Closing the file

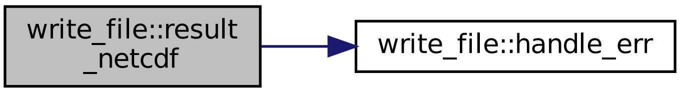
Definition at line 19 of file netcdf_file.f90.

```
    INTEGER, PARAMETER :: dp=kind(1.0d0)
    CHARACTER(LEN=100) :: filename = 'results.nc4'
    REAL(dp), DIMENSION(:), INTENT(IN) :: dof
    REAL(dp), DIMENSION(:), INTENT(IN) :: ele
    INTEGER, INTENT(IN) :: num_ele, num_nuc

    CHARACTER(LEN=100), DIMENSION(1) :: dim_dof = (/"Optimal_DOF"/)
    CHARACTER(LEN=100), DIMENSION(1) :: dim_ele = (/"Electron_Density"/)

    ! File ID's
    INTEGER :: ierr, file_id
    ! Variable ID's
    INTEGER :: var_dof,  var_ele, var_num_ele, var_num_nuc
    ! Dimension ID's for the arrays
    INTEGER, DIMENSION(1) :: did_dof, did_ele
    ! Implicit types for the sizes for the arrays
    INTEGER, DIMENSION(1) :: size_dof, size_ele

    ! Create the file
    ierr = nf90_create(filename, nf90_clobber, file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Variable sizes
    size_dof = shape(dof)
    size_ele = shape(ele)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)


    ! Number of electrons
    ierr = nf90_def_var(file_id, "Num_of_Electrons", nf90_double, var_num_ele)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Number of nuclei
    ierr = nf90_def_var(file_id, "Num_of_Nuclei", nf90_double, var_num_nuc)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! DOF
    ierr = nf90_def_dim(file_id, dim_dof(1), size_dof(1), did_dof(1))
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    ierr = nf90_def_var(file_id, "Optimal_DOF", nf90_double, did_dof, var_dof)

    ! Electron Density
    ierr = nf90_def_dim(file_id, dim_ele(1), size_ele(1), did_ele(1))
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    ierr = nf90_def_var(file_id, "Electron_Density", nf90_double, did_ele, var_ele)


    ! Metadata
    ierr = nf90_enddef(file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)



    ! Number of electrons
    ierr = nf90_put_var(file_id, var_num_ele, num_ele)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Number of nuclei
    ierr = nf90_put_var(file_id, var_num_nuc, num_nuc)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! DOF
    ierr = nf90_put_var(file_id, var_dof, dof)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ! Electron Density
    ierr = nf90_put_var(file_id, var_ele, ele)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)


     ierr = nf90_close(file_id)
     IF (ierr /= nf90_noerr) CALL handle_err(ierr)

     !Finishing statement if writing file is successful.
     print *, "Success in writing the NETCDF file: ", filename

```

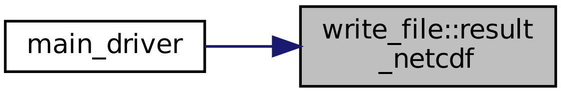
References handle_err().

Referenced by main_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



11.18.2.4 write_restart_file()

```
subroutine write_file::write_restart_file (
            real(dp) gp_n_data,
            real(dp) gp_n_dof,
            real(dp) n_cycles,
            real(dp) no_samples,
            real(dp) current_best_E,
            real(dp) constant_mean_value,
            real(dp) gamma,
            real(dp) kernel_var,
            real(dp) kernal_inv_length,
            real(dp), dimension(:), intent(in) E_data,
            real(dp), dimension(:), intent(in) data_mean,
            real(dp), dimension(:,:), intent(in) param_pres,
            real(dp), dimension(:,:), intent(in) param_cov,
            real(dp), dimension(:,:), intent(in) param_data )
```

Creates a NetCDF file which contains information needed for the restart of the Bi_Op_step function.

This file will be created in the event that the Bi_Op_step function crashes.

Parameters

	gp_n_data	Defining everything for the scalar variables
	gp_n_data	is the data used for the GP surrogate
	gp_n_dof	is the degrees of freedom used for the GP surrogate
	n_cycles	is the number of cycles
	no_samples	is the number of samples
	current_best_E	is the current best energy found so far during the run.
	constant_mean_value	
	gamma	
	kernel_var	
	kernal_inv_length	
	gp_n_dof	Defining everything for the scalar variables
	gp_n_data	is the data used for the GP surrogate
	gp_n_dof	is the degrees of freedom used for the GP surrogate
	n_cycles	is the number of cycles

Parameters

	no_samples	is the number of samples
	current_best_E	is the current best energy found so far during the run.
	constant_mean_value	
	gamma	
	kernel_var	
	kernal_inv_length	
	n_cycles	Defining everything for the scalar variables
	gp_n_data	is the data used for the GP surrogate
	gp_n_dof	is the degrees of freedom used for the GP surrogate
	n_cycles	is the number of cycles
	no_samples	is the number of samples
	current_best_E	is the current best energy found so far during the run.
	constant_mean_value	
	gamma	
	kernel_var	
	kernal_inv_length	
	no_samples	Defining everything for the scalar variables
	gp_n_data	is the data used for the GP surrogate
	gp_n_dof	is the degrees of freedom used for the GP surrogate
	n_cycles	is the number of cycles
	no_samples	is the number of samples
	current_best_E	is the current best energy found so far during the run.
	constant_mean_value	
	gamma	
	kernel_var	
	kernal_inv_length	
	current_best_e	Defining everything for the scalar variables
	gp_n_data	is the data used for the GP surrogate
	gp_n_dof	is the degrees of freedom used for the GP surrogate
	n_cycles	is the number of cycles
	no_samples	is the number of samples
	current_best_E	is the current best energy found so far during the run.
	constant_mean_value	
	gamma	
	kernel_var	
	kernal_inv_length	
`in`	*e_data*	Defining everything for the array variables Rank 1 arrays
	E_data	is a rank 1 array containing the energy data
	data_mean	is the
`in`	*param_pres*	Rank 2 arrays
	param_pres	rank 2 array containing @param_cov rank 2 array containing the covariance matrix
	param_data	

Dimension Names - named the same as the input variables

ID's

Create the file

Defining the Scalar Variables

Inputting in the Array variables and defining the dimensions

Rank 1 arrays first

Rank 2 arrays

Writing the array data to file

Scalar Values

Rank 1 arrays

Rank 2 arrays

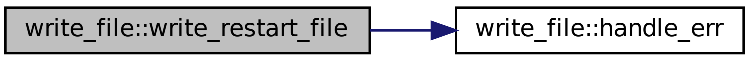
Closing the file

Definition at line 224 of file netcdf_file.f90.

```


    INTEGER, PARAMETER :: dp=kind(1.0d0)
    ! Declaring the input variables
    CHARACTER(LEN=100) :: filename = 'restart.nc4'

    INTEGER :: ierr, file_id, i

    REAL(dp) :: gp_n_data, gp_n_dof, n_cycles, no_samples, current_best_E
    REAL(dp) :: constant_mean_value, gamma, kernel_var, kernal_inv_length

    INTEGER :: var_gp_n_data, var_gp_n_dof, var_n_cycles, var_no_samples
    INTEGER :: var_current_best_E, var_constant_mean_value, var_gamma
    INTEGER :: var_kernel_var, var_kernal_inv_length

    REAL(dp), DIMENSION(:), INTENT(IN) :: E_data
    REAL(dp), DIMENSION(:), INTENT(IN) :: data_mean


    REAL(dp), DIMENSION(:,:), INTENT(IN) :: param_pres
    REAL(dp), DIMENSION(:,:), INTENT(IN) :: param_cov
    REAL(dp), DIMENSION(:,:), INTENT(IN) :: param_data

    CHARACTER(LEN=100), DIMENSION(1) :: dim_e_data = (/"E_Data"/)
    CHARACTER(LEN=100), DIMENSION(1) :: dim_data_mean = (/"data_mean"/)

    CHARACTER(LEN=100), DIMENSION(2) :: dim_param_pres = (/"param_pres","param_pres"/)
    CHARACTER(LEN=100), DIMENSION(2) :: dim_param_cov = (/"param_cov","param_cov"/)
    CHARACTER(LEN=100), DIMENSION(2) :: dim_param_data = (/"param_data","param_data"/)

    ! Variable IDs
    INTEGER :: var_e_data, var_data_mean, var_param_pres
    INTEGER :: var_param_cov, var_param_data

    ! Dimension IDs for the arrays
    INTEGER, DIMENSION(1) :: did_e_data, did_data_mean
    INTEGER, DIMENSION(2) :: did_param_cov, did_param_data, did_param_pres

    ! Sizes for the arrays
    INTEGER, DIMENSION(1) :: size_e_data, size_data_mean
    INTEGER, DIMENSION(2) :: size_param_cov, size_param_data, size_param_pres


    ierr = nf90_create(filename, nf90_clobber, file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)


    ! Variable sizes
    size_e_data = shape(e_data)
    size_data_mean = shape(data_mean)
    size_param_pres = shape(param_pres)
    size_param_cov = shape(param_cov)
    size_param_data = shape(param_data)


    ierr = nf90_def_var(file_id, "gp_n_data", nf90_double, var_gp_n_data)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "gp_n_dof", nf90_double, var_gp_n_dof)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "n_cycles", nf90_double, var_n_cycles)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "no_samples", nf90_double, var_no_samples)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "current_best_e", nf90_double, var_current_best_e)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "constant_mean_value", nf90_double, var_constant_mean_value)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "gamma", nf90_double, var_gamma)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "kernel_var", nf90_double, var_kernel_var)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_def_var(file_id, "kernal_inv_length", nf90_double, var_kernal_inv_length)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)




    ierr = nf90_def_dim(file_id, dim_e_data(1), size_e_data(1), did_e_data(1))
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    ierr = nf90_def_var(file_id, "E_data", nf90_double, did_e_data, var_e_data)

    ierr = nf90_def_dim(file_id, dim_data_mean(1), size_data_mean(1), did_data_mean(1))
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    ierr = nf90_def_var(file_id, "data_mean", nf90_double, did_data_mean, var_data_mean)


    DO i = 1,2
      ierr = nf90_def_var(file_id, dim_param_pres(i), size_param_pres(i), did_param_pres(i))
      IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    END DO
    ierr = nf90_def_var(file_id, "param_pres", nf90_double, did_param_pres, var_param_pres)

    DO i = 1,2
      ierr = nf90_def_var(file_id, dim_param_cov(i), size_param_cov(i), did_param_cov(i))
      IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    END DO
    ierr = nf90_def_var(file_id, "param_cov", nf90_double, did_param_cov, var_param_cov)

    DO i = 1,2
      ierr = nf90_def_var(file_id, dim_param_data(i), size_param_data(i), did_param_data(i))
      IF (ierr /= nf90_noerr) CALL handle_err(ierr)
    END DO
    ierr = nf90_def_var(file_id, "param_data", nf90_double, did_param_data, var_param_data)


    ! Metadata
    ierr = nf90_enddef(file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)




    ierr = nf90_put_var(file_id, var_gp_n_data, gp_n_data)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_gp_n_dof, gp_n_dof)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_n_cycles, n_cycles)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_no_samples, no_samples)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_current_best_e, current_best_e)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_constant_mean_value, constant_mean_value)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_gamma, gamma)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_kernel_var, kernel_var)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_kernal_inv_length, kernal_inv_length)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)


    ierr = nf90_put_var(file_id, var_e_data, e_data)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_data_mean, data_mean)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)


    ierr = nf90_put_var(file_id, var_param_pres, param_pres)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_param_cov, param_cov)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    ierr = nf90_put_var(file_id, var_param_data, param_data)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)



    ierr = nf90_close(file_id)
    IF (ierr /= nf90_noerr) CALL handle_err(ierr)

    print *, "Success in writing file: ", filename

```

References handle_err().

Here is the call graph for this function:



11.18.2.5 xy_grid()

```
subroutine write_file::xy_grid (
            integer, intent(in) points,
            real(dp), intent(in) box_size )
```

Writes the xyz coordinates used in the calculation of either the wavefunction or electron density to a text file names xyz.txt.

This only produces a equally spaced square grid of points in the x-y plane, with the z coordinate being set to zero. These coordinates are used for the 2D contour plot in the resulting output plot.

Parameters

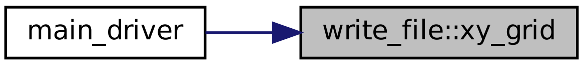
in	*box_size*	box_size is the user defined parameter which defines the size of the domain in each axis.
in	*points*	points is the user defined

Definition at line 185 of file netcdf_file.f90.

```

    INTEGER, PARAMETER :: dp=kind(1.0d0)
    REAL(dp), INTENT(IN) :: box_size
    INTEGER, INTENT(IN) :: points
    ! Loop variables
    INTEGER :: i,j
    ! Resulting position values in x-y axis
    REAL(dp) :: a, b ! a is x, b is y
    INTEGER, PARAMETER :: file_no=3

    ! Opens an empty text file
    OPEN (unit=file_no,file="xyz.txt",action="write")
    DO i = -points, points
      DO j = -points, points
        ! Scales the positions to be between -box_size and +box_size
        a =  (real(i) / (abs(points) + abs(points)))*box_size
        b =  (real(j) / (abs(points) + abs(points)))*box_size
        ! Writes the coordinates to file, the z coordinate is set to zero as only using this for 2D
   contour plot which requires x and y coordinates to change.
        write (file_no,*) a,b,0
      END DO
    END DO
    ! Closes the file after writing
    CLOSE (file_no)
    print*, 'Success in writing: xyz.txt'
```

Referenced by main_driver().

Here is the caller graph for this function:



Chapter 12

Data Type Documentation

12.1 biased_optim::bi_op_init Interface Reference

Public Member Functions

- subroutine bi_op_init_constant_mean (param_init_data, energy_init_data, n_data_in, n_dof_in, ker_var, ker_lengthscale, constant_mean_prior, optim_rate_para, optim_no_samples, n_threads, n_loops_to_do)
- subroutine bi_op_init_arb_mean (param_init_data, energy_init_data, n_data_in, n_dof_in, ker_var, ker_↩ lengthscale, mean_prior_func, mean_prior_dx, optim_rate_para, optim_no_samples, n_threads, n_loops_↩ to_do)

12.1.1 Detailed Description

Definition at line 17 of file Biased_Optim.f90.

12.1.2 Member Function/Subroutine Documentation

12.1.2.1 bi_op_init_arb_mean()

```
subroutine biased_optim::bi_op_init::bi_op_init_arb_mean (
            real(dp), dimension(n_data_in, n_dof_in), intent(in) param_init_data,
            real(dp), dimension(n_data_in), intent(in) energy_init_data,
            integer, intent(in) n_data_in,
            integer, intent(in) n_dof_in,
            real(dp), intent(in) ker_var,
            real(dp), intent(in) ker_lengthscale,
            procedure(mean_func_interface) mean_prior_func,
            procedure(mean_func_interface_dx) mean_prior_dx,
            real(dp), intent(in) optim_rate_para,
            integer, intent(in) optim_no_samples,
            integer, intent(in) n_threads,
            integer, intent(inout) n_loops_to_do )
```

Definition at line 134 of file Biased_Optim.f90.

```
        implicit none
        integer, intent(inout) :: n_loops_to_do
        integer, intent(in) :: n_data_in, n_dof_in, optim_no_samples,n_threads
        real(dp), dimension(n_data_in, n_dof_in), intent(in) :: param_init_data
        real(dp), dimension(n_data_in), intent(in) :: energy_init_data
        real(dp), intent(in) :: ker_var, ker_lengthscale,  optim_rate_para
        procedure(mean_func_interface) :: mean_prior_func
        procedure(mean_func_interface_dx) :: mean_prior_dx

        n_dof = n_dof_in
        gamma = optim_rate_para
        no_samples=optim_no_samples

        call gp_init(mean_prior_func, mean_prior_dx, ker_var, ker_lengthscale, param_init_data,
energy_init_data,&
         n_data_in, n_dof_in, n_threads)
        gp_uptodate = .false.

        current_best_e = minval(energy_init_data)

```

12.1.2.2 bi_op_init_constant_mean()

```
subroutine biased_optim::bi_op_init::bi_op_init_constant_mean (
              real(dp), dimension(n_data_in, n_dof_in), intent(in) param_init_data,
              real(dp), dimension(n_data_in), intent(in) energy_init_data,
              integer, intent(in) n_data_in,
              integer, intent(in) n_dof_in,
              real(dp), intent(in) ker_var,
              real(dp), intent(in) ker_lengthscale,
              real(dp), intent(in) constant_mean_prior,
              real(dp), intent(in) optim_rate_para,
              integer, intent(in) optim_no_samples,
              integer, intent(in) n_threads,
              integer, intent(inout) n_loops_to_do )
```

Definition at line 84 of file Biased_Optim.f90.

```
        implicit none
        integer, intent(inout) :: n_loops_to_do
        integer, intent(in) :: n_data_in, n_dof_in, optim_no_samples,n_threads
        real(dp), dimension(n_data_in, n_dof_in), intent(in) :: param_init_data
        real(dp), dimension(n_data_in), intent(in) :: energy_init_data
        real(dp), intent(in) :: ker_var, ker_lengthscale, constant_mean_prior, optim_rate_para
        integer gp_n_data, gp_n_dof, n_cycles
        real(dp) :: kernel_var, kernal_inv_length
        real(dp), dimension(:,:), allocatable :: param_data
        real(dp), dimension(:), allocatable :: E_data
        real(dp), dimension(:,:), allocatable :: param_pres, param_cov
        real(dp), dimension(:), allocatable :: data_mean

        if (find_restart_file()) then
            call read_restart_file_sizes(gp_n_data, gp_n_dof)
            allocate(param_data(gp_n_data, gp_n_dof))
             allocate(e_data(gp_n_data))
             allocate(param_pres(gp_n_data, gp_n_data))
             allocate(param_cov(gp_n_data, gp_n_data))
             allocate(data_mean(gp_n_data))
             call read_restart_file_data(n_cycles, no_samples, current_best_e,&
             constant_mean_value, kernel_var, gamma, kernal_inv_length,&
             gp_n_data, e_data, gp_n_data, data_mean, gp_n_data, gp_n_data, param_pres, &
             gp_n_data, gp_n_data, param_pres, gp_n_data, gp_n_dof, param_data)
             n_dof = gp_n_dof

             call gp_restart(gp_n_data, gp_n_dof,kernel_var, kernal_inv_length,&
             param_data, e_data,param_pres, param_cov, data_mean, constant_mean,&
              zero_func, n_threads)
             gp_uptodate = .false.
             current_best_e = minval(e_data)
             n_loops_to_do=n_loops_to_do-(n_cycles-1)
         else
             constant_mean_value = constant_mean_prior
             gamma = optim_rate_para
             no_samples=optim_no_samples
             n_dof = n_dof_in

             call gp_init(constant_mean, zero_func, ker_var, ker_lengthscale, param_init_data,
    energy_init_data, n_data_in,&
              n_dof_in,n_threads)
             gp_uptodate = .false.

             current_best_e = minval(energy_init_data)
         end if

```

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

- Biased_Optim.f90

12.2 gp_surrogate::cov_kernal_dx_1_interface Interface Reference

Public Member Functions

- real(dp) function cov_kernal_dx_1_interface (x_1, x_2, dim)

12.2.1 Detailed Description

Definition at line 57 of file GP_surrogate.f90.

12.2.2 Constructor & Destructor Documentation

12.2.2.1 cov_kernal_dx_1_interface()

```
real(dp) function gp_surrogate::cov_kernal_dx_1_interface::cov_kernal_dx_1_interface (
              real(dp), dimension(:), intent(in) x_1,
              real(dp), dimension(:), intent(in) x_2,
              integer, intent(in) dim )
```

Definition at line 58 of file GP_surrogate.f90.

```
           use shared_constants
           implicit none
           integer, intent(in) :: dim
           real(dp), dimension(:), intent(in) :: x_1 !param space point
           real(dp), dimension(:), intent(in) :: x_2 !param space point
           real(dp) :: out
```

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

- GP_surrogate.f90

12.3 gp_surrogate::cov_kernal_interface Interface Reference

Public Member Functions

- real(dp) function cov_kernal_interface (x_1, x_2)

12.3.1 Detailed Description

Definition at line 46 of file GP_surrogate.f90.

12.3.2 Constructor & Destructor Documentation

12.3.2.1 cov_kernal_interface()

```
real(dp) function gp_surrogate::cov_kernal_interface::cov_kernal_interface (
              real(dp), dimension(:), intent(in) x_1,
              real(dp), dimension(:), intent(in) x_2 )
```

Definition at line 47 of file GP_surrogate.f90.

```
              use shared_constants
              implicit none
              real(dp), dimension(:), intent(in) :: x_1 !param space point
              real(dp), dimension(:), intent(in) :: x_2 !param space point
              real(dp) :: out
```

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

- GP_surrogate.f90

12.4 gp_surrogate::cov_kernal_xx_dx_interface Interface Reference

Public Member Functions

- real(dp) function cov_kernal_xx_dx_interface (x, dim)

12.4.1 Detailed Description

Definition at line 68 of file GP_surrogate.f90.

12.4.2 Constructor & Destructor Documentation

12.4.2.1 cov_kernal_xx_dx_interface()

```
real(dp) function gp_surrogate::cov_kernal_xx_dx_interface::cov_kernal_xx_dx_interface (
              real(dp), dimension(:), intent(in) x,
              integer, intent(in) dim )
```

Definition at line 69 of file GP_surrogate.f90.

```
              use shared_constants
              implicit none
              integer, intent(in) :: dim
              real(dp), dimension(:), intent(in) :: x !param space point
              real(dp) :: out
```

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

- GP_surrogate.f90

12.5 gp_surrogate::gp_init Interface Reference

general initialisation function, see gp_init_gausscov

Public Member Functions

- subroutine gp_init_arbcov (mean_prior, cov_prior, param_data_in, E_data_in, n_data_in, n_dof_in)

  do not use, don't have a function for the dervatives of the kernal/mean
- subroutine gp_init_gausscov (mean_prior, mean_prior_dx, ker_var, ker_length, param_data_in, E_data_in, n_data_in, n_dof_in, n_threads_in)

  intialises with a gaussian covariance

12.5.1 Detailed Description

general initialisation function, see gp_init_gausscov

Definition at line 86 of file GP_surrogate.f90.

12.5.2 Member Function/Subroutine Documentation

12.5.2.1 gp_init_arbcov()

```
subroutine gp_surrogate::gp_init::gp_init_arbcov (
            procedure(mean_func_interface) mean_prior,
            procedure(cov_kernal_interface) cov_prior,
            real(dp), dimension(n_data_in,n_dof_in), intent(in) param_data_in,
            real(dp), dimension(n_data_in), intent(in) E_data_in,
            integer, intent(in) n_data_in,
            integer, intent(in) n_dof_in )
```

do not use, don't have a function for the dervatives of the kernal/mean

Definition at line 194 of file GP_surrogate.f90.

```
        implicit none
        procedure(mean_func_interface) :: mean_prior
        procedure(cov_kernal_interface) :: cov_prior
        integer, intent(in) :: n_data_in, n_dof_in
        real(dp), dimension(n_data_in,n_dof_in), intent(in) :: param_data_in
        real(dp), dimension(n_data_in), intent(in) :: E_data_in
        !work and ipiv are needed for lapack call, have no useful info
        integer :: i,j

        !setting priors
        mu_prior => mean_prior
        k_prior => cov_prior

        !setting data
        n_data = n_data_in
        n_dof = n_dof_in
        allocate(param_data(n_data,n_dof))
        allocate(e_data(n_data))
        e_data = e_data_in
        param_data = param_data_in


        allocate(param_cov(n_data,n_data))
        allocate(param_pres(n_data,n_data))

        !finds covarience for new data
        do i=1,n_data
            do j=1,i
                !is symmteric, this reduces calls
                param_cov(i,j) = k_prior(param_data(i,:), param_data(j,:))
                param_cov(j,i) = param_cov(i,j)
            end do
        end do

        !compute inverse of cov (precsion)
        param_pres = param_cov
        call svd_inverse(param_pres,  n_data)

        allocate(data_mean(n_data))
        !updating the prior mean list
        do i=1,n_data
            data_mean(i) = mu_prior(param_data(i,:))
        end do

        init = .true.

```

12.5.2.2 gp_init_gausscov()

```
subroutine gp_surrogate::gp_init::gp_init_gausscov (
              procedure(mean_func_interface) mean_prior,
              procedure(mean_func_interface_dx) mean_prior_dx,
              real(dp), intent(in) ker_var,
              real(dp), intent(in) ker_length,
              real(dp), dimension(n_data_in,n_dof_in), intent(in) param_data_in,
              real(dp), dimension(n_data_in), intent(in) E_data_in,
              integer, intent(in) n_data_in,
              integer, intent(in) n_dof_in,
              integer, intent(in) n_threads_in )
```

intialises with a gaussian covariance

Definition at line 133 of file GP_surrogate.f90.

```
        implicit none
        procedure(mean_func_interface) :: mean_prior
        procedure(mean_func_interface_dx) :: mean_prior_dx
        integer, intent(in) :: n_data_in, n_dof_in,n_threads_in
        real(dp), intent(in) :: ker_var, ker_length
        real(dp), dimension(n_data_in,n_dof_in), intent(in) :: param_data_in
        real(dp), dimension(n_data_in), intent(in) :: E_data_in
        integer :: i,j

        n_threads=n_threads_in

        !setting prior parameters
        kernel_var = ker_var
        kernal_inv_length = 1.0_dp/ker_length
        !setting priors
        mu_prior => mean_prior
        k_prior => gaussian_kernel

        !derivatives of priors, needed for SGA step later
        mu_prior_dx => mean_prior_dx
        k_prior_dx_1_i => gaussian_kernel_dx_1_i
        k_prior_xx_dx_i => zero_func
        gp_k_post_xx_d_x_i => zero_func !prior 0=>post 0, this saves having to evaluate (or write, until
     support is expanded) code to get the 0

        !setting data
        n_data = n_data_in
        n_dof = n_dof_in
        allocate(param_data(n_data,n_dof))
        allocate(e_data(n_data))
        e_data = e_data_in
        param_data = param_data_in


        allocate(param_cov(n_data,n_data))
        allocate(param_pres(n_data,n_data))

        !finds covarience for new data
        do i=1,n_data
            do j=1,i
                !is symmteric, this reduces calls
                param_cov(i,j) = k_prior(param_data(i,:), param_data(j,:))
                param_cov(j,i) = param_cov(i,j)
            end do
        end do

        !compute inverse of cov (precsion)
        param_pres = param_cov
        call svd_inverse(param_pres,  n_data)

        allocate(data_mean(n_data))
        !updating the prior mean list
        do i=1,n_data
            data_mean(i) = mu_prior(param_data(i,:))
        end do

        init = .true.

```

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

- GP_surrogate.f90

12.6 gp_surrogate::gp_k_post Interface Reference

function for postieror covairance kernal use in format x_1, x_2, x1_dim, x2_dim for cov(x_1,x_2) use in format x, x_dim for cov(x,x)

Public Member Functions

- real(dp) function, dimension(x1_dim, x2_dim) gp_k_post_diff_x (x1, x2, x1_dim, x2_dim)

  specific function for cov(x_1,x_2), see gp_k_post
- real(dp) function, dimension(x_dim, x_dim) gp_k_post_same_x (x, x_dim)

  specific function for cov(x,x), see gp_k_post

12.6.1 Detailed Description

function for postieror covairance kernal use in format x_1, x_2, x1_dim, x2_dim for cov(x_1,x_2) use in format x, x_dim for cov(x,x)

Definition at line 81 of file GP_surrogate.f90.

12.6.2 Member Function/Subroutine Documentation

12.6.2.1 gp_k_post_diff_x()

```
real(dp) function, dimension(x1_dim,x2_dim) gp_surrogate::gp_k_post::gp_k_post_diff_x (
              real(dp), dimension(x1_dim,n_dof), intent(in) x1,
              real(dp), dimension(x2_dim,n_dof), intent(in) x2,
              integer, intent(in) x1_dim,
              integer, intent(in) x2_dim )
```

specific function for cov(x_1,x_2), see gp_k_post

Definition at line 356 of file GP_surrogate.f90.

```
        implicit none
        integer, intent(in) :: x1_dim, x2_dim
        real(dp), dimension(x1_dim,n_dof), intent(in) :: x1 !param space point
        real(dp), dimension(x2_dim,n_dof), intent(in) :: x2 !param space point
        real(dp), dimension(x1_dim,x2_dim) :: out, x1_cov_x2
        real(dp), dimension(n_data,x2_dim) :: data_cov_x2
        real(dp), dimension(x1_dim,n_data) :: x1_cov_data
        integer :: i,j

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        do i=1,x2_dim
            do j=1,n_data
                data_cov_x2(j,i) = k_prior(x2(i,:), param_data(j,:))
            end do
        end do

        do i=1,x1_dim
            do j=1,n_data
                x1_cov_data(i,j) = k_prior(param_data(j,:),x1(i,:))
            end do
        end do

        do i=1,x1_dim
            do j=1,x2_dim
                x1_cov_x2(i,j) = k_prior(x1(i,:), x2(j,:))
            end do
        end do

       out = x1_cov_x2 + matmul(matmul(x1_cov_data,param_pres),data_cov_x2)

```

12.6.2.2 gp_k_post_same_x()

```
real(dp) function, dimension(x_dim,x_dim) gp_surrogate::gp_k_post::gp_k_post_same_x (
              real(dp), dimension(x_dim,n_dof), intent(in) x,
              integer, intent(in) x_dim )
```

specific function for cov(x,x), see gp_k_post

Definition at line 322 of file GP_surrogate.f90.

```
        implicit none
        integer, intent(in) :: x_dim
        real(dp), dimension(x_dim,n_dof), intent(in) :: x !param space point
        real(dp), dimension(x_dim,x_dim) :: out, x_cov_x
        real(dp), dimension(x_dim,n_data) :: x_cov_data
        real(dp), dimension(n_data,x_dim) :: data_cov_x
        integer :: i,j

        if (.not. init) then
            print*,'not intialised'
            stop
        end if

        do i=1,x_dim
            do j=1,n_data
                data_cov_x(j,i) = k_prior(param_data(j,:), x(i,:))
            end do
        end do

        x_cov_data = transpose(data_cov_x)
        do i=1,x_dim
            do j=1,i
                !is symmteric, this reduces calls
                x_cov_x(i,j) = k_prior(x(i,:), x(j,:))
                x_cov_x(j,i) = x_cov_x(i,j)
            end do
        end do

       out = x_cov_x + matmul(matmul(x_cov_data,param_pres),data_cov_x)

```

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

- GP_surrogate.f90

12.7 mcmc::log_rho_interface Interface Reference

Public Member Functions

- real(dp) function log_rho_interface (x, dof)

12.7.1 Detailed Description

Definition at line 17 of file MCMC.f90.

12.7.2 Constructor & Destructor Documentation

12.7.2.1 log_rho_interface()

```
real(dp) function mcmc::log_rho_interface::log_rho_interface (
            real(dp), dimension(:), intent(in) x,
            real(dp), dimension(:), intent(in) dof )
```

Definition at line 18 of file MCMC.f90.

```
       !log target dist
       use shared_constants
       real(dp), dimension(:), intent(in) :: x
       real(dp), dimension(:), intent(in) :: dof
       real(dp) :: out
```

References shared_constants::pi.

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

- MCMC.f90

12.8 gp_surrogate::mean_func_interface Interface Reference

Public Member Functions

- real(dp) function mean_func_interface (x)

12.8.1 Detailed Description

Definition at line 26 of file GP_surrogate.f90.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 mean_func_interface()

```
real(dp) function gp_surrogate::mean_func_interface::mean_func_interface (
            real(dp), dimension(:), intent(in) x )
```

Definition at line 27 of file GP_surrogate.f90.

```
            use shared_constants
            implicit none
            real(dp), dimension(:), intent(in) :: x !param space point
            real(dp) :: out
```

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

- GP_surrogate.f90

12.9 gp_surrogate::mean_func_interface_dx Interface Reference

Public Member Functions

- real(dp) function mean_func_interface_dx (x, dim)

12.9.1 Detailed Description

Definition at line 35 of file GP_surrogate.f90.

12.9.2 Constructor & Destructor Documentation

12.9.2.1 mean_func_interface_dx()

```
real(dp) function gp_surrogate::mean_func_interface_dx::mean_func_interface_dx (
              real(dp), dimension(:), intent(in) x,
              integer, intent(in) dim )
```

Definition at line 36 of file GP_surrogate.f90.

```
          use shared_constants
          implicit none
          integer, intent(in) :: dim !dimension along which differentiation occurs
          real(dp), dimension(:), intent(in) :: x !param space point
          real(dp) :: out
```

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

- GP_surrogate.f90

12.10 basis_functions::wave_function_interface Interface Reference

Public Member Functions

- real(dp) function wave_function_interface (position, dof_coefficients)

12.10.1 Detailed Description

Definition at line 25 of file basis_functions.f90.

12.10.2 Constructor & Destructor Documentation

12.10.2.1 wave_function_interface()

```
real(dp) function basis_functions::wave_function_interface::wave_function_interface (
              real(dp), dimension(:), intent(in) position,
              real(dp), dimension(:), intent(in) dof_coefficients )
```

Definition at line 26 of file basis_functions.f90.

```
        use shared_constants
        real(dp), dimension(:), intent(in) :: position
        real(dp), dimension(:), intent(in) :: dof_coefficients
        real(dp) :: wave_function_interface
```

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

- basis_functions.f90

Chapter 13

File Documentation

13.1 acknow.txt File Reference

13.2 basis_functions.f90 File Reference

Functions that define the Hamiltonian that specifies the problem and the basis set being used.

Data Types

- interface basis_functions::wave_function_interface

Modules

- module basis_functions

  Wavefunction, hamiltonian and basis set choice.

Functions/Subroutines

- subroutine basis_functions::initialise_basis (n_electrons_in, n_basis_functions_per_atom_in, n_atoms_in, atom_coords_in, n_Jastrow_in, fd_length_in, Jastrow_b_length_in, Jastrow_d_length_in, proton_numbers↩ _in, basis_type_in)

  Initialisation routine.
- subroutine basis_functions::deinitialise_basis

  Dinitialisation routine.
- real(dp) function basis_functions::wave_function_slater_1s (position, dof_coefficients)

  Single Electron wavefunction: slater 1s basis.
- real(dp) function basis_functions::reduced_hamiltonian_slater_1s (position, dof_coefficients)

  Single Electron Reduced Hamiltonian: slater 1s basis.
- real(dp) function basis_functions::discrete_laplacian_reduced (position, h, dofs)

  Finite Difference Reduced Laplacian Computes finite difference approximation to the reduced laplacian $(\nabla^2\psi)/\psi$.
- real(dp) function basis_functions::wave_function_2_electrons (position, dof_coefficients)

  Wavefunction for 2 electrons.

- real(dp) function basis_functions::reduced_hamiltonian_2_electrons (position, dof_coefficients)

  Reduced Hamiltonian for 2 electrons.
- subroutine basis_functions::mno_allocate (n_terms)

  Allocate parameters for Jastrow factor.
- real(dp) function basis_functions::log_density (position, dof_coefficients)

  Log of the Probability Density.
- real(dp) function basis_functions::wave_function_sto3g (position, dof_coefficients)

  Single Electron wavefunction: gaussian sto3g basis.
- real(dp) function basis_functions::reduced_hamiltonian_sto3g (position, dof_coefficients)

  Single Electron Reduced Hamiltonian: gaussian sto3g basis.

Variables

- procedure(wave_function_interface), pointer basis_functions::wave_function
- procedure(wave_function_interface), pointer basis_functions::reduced_hamiltonian
- logical, protected basis_functions::initialised = .false.
- integer, protected basis_functions::n_electrons
- integer, protected basis_functions::n_basis_functions_per_atom
- integer, protected basis_functions::n_atoms
- real(dp), dimension(:,:), allocatable, protected basis_functions::atom_coords
- integer, protected basis_functions::n_jastrow_dofs
- real(dp), protected basis_functions::fd_h
- real(dp), protected basis_functions::b_length
- real(dp), protected basis_functions::d_length
- real(dp), dimension(:), allocatable, protected basis_functions::proton_numbers
- integer, protected basis_functions::basis_type
- integer, protected basis_functions::number_dofs
- integer, protected basis_functions::n_space_dims
- real(dp), dimension(:,:), allocatable, protected basis_functions::dof_bounds
- integer, protected basis_functions::n_dofs_per_atom
- integer, protected basis_functions::n_dofs_no_jastrow
- integer, dimension(:,:), allocatable, protected basis_functions::mno_parameters
- procedure(wave_function_interface), pointer basis_functions::wave_function_single

13.2.1 Detailed Description

Functions that define the Hamiltonian that specifies the problem and the basis set being used.

13.3 bias_opt.txt File Reference

13.4 Biased_Optim.f90 File Reference

Biased optimization subroutines and functions.

Data Types

- interface biased_optim::bi_op_init

Modules

- module biased_optim

  Biased optimization subroutines and functions.

Functions/Subroutines

- logical function biased_optim::find_restart_file ()
- subroutine biased_optim::bi_op_init_constant_mean (param_init_data, energy_init_data, n_data_in, n_dof↩_in, ker_var, ker_lengthscale, constant_mean_prior, optim_rate_para, optim_no_samples, n_threads, n_↩loops_to_do)
- subroutine biased_optim::bi_op_init_arb_mean (param_init_data, energy_init_data, n_data_in, n_dof_in, ker_var, ker_lengthscale, mean_prior_func, mean_prior_dx, optim_rate_para, optim_no_samples, n_threads, n_loops_to_do)
- real(dp) function, dimension(threads, n_dof) biased_optim::bi_op_step (param_update_data, energy_↩update_data, n_new_data, threads, seed, n_cycles)

Variables

- logical biased_optim::gp_uptodate = .False.

13.4.1 Detailed Description

Biased optimization subroutines and functions.

13.5 Biased_Optim_example_driver.f90 File Reference

Driver for testng the biased optimization routines.

Modules

- module log_rho_mod

  Driver for testng the biased optimization routines.

Functions/Subroutines

- real(dp) function log_rho_mod::log_rho (x, dof)
- program main

13.5.1 Detailed Description

Driver for testng the biased optimization routines.

13.5.2 Function/Subroutine Documentation

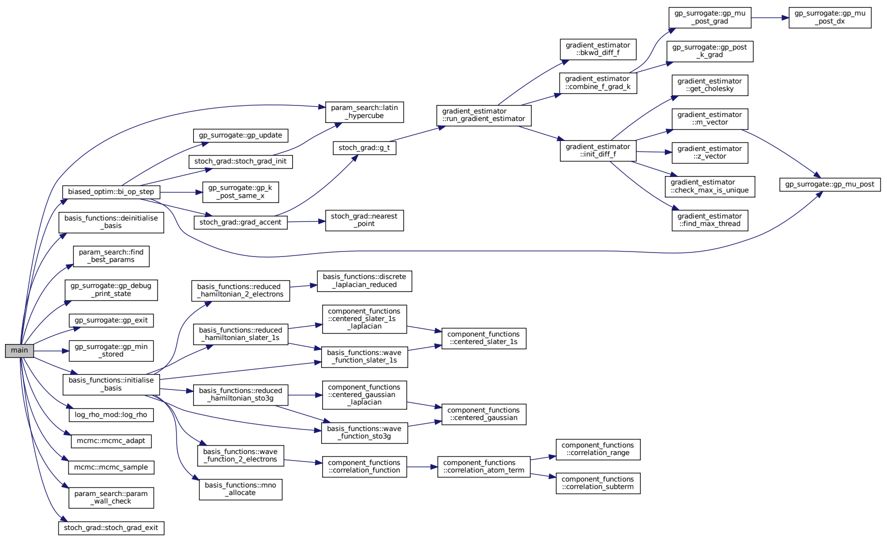
13.5.2.1 main()

```
program main
```

Definition at line 23 of file Biased_Optim_example_driver.f90.

References param_search::best_params, param_search::best_trial, biased_optim::bi_op_step(), basis_functions↩ ::deinitialise_basis(), basis_functions::dof_bounds, shared_constants::dp, shared_constants::electronvolt, param↩ _search::find_best_params(), gp_surrogate::gp_debug_print_state(), gp_surrogate::gp_exit(), gp_surrogate↩ ::gp_min_stored(), basis_functions::initialise_basis(), param_search::latin_hypercube(), log_rho_mod::log_rho(), mcmc::mcmc_adapt(), mcmc::mcmc_sample(), basis_functions::number_dofs, param_search::param_wall_↩ check(), basis_functions::reduced_hamiltonian, and stoch_grad::stoch_grad_exit().

Here is the call graph for this function:


main
biased_optim::bi_op_step
basis_functions::deinitialise_basis
param_search::find_best_params
gp_surrogate::gp_debug_print_state
gp_surrogate::gp_exit
gp_surrogate::gp_min_stored
basis_functions::initialise_basis
log_rho_mod::log_rho
mcmc::mcmc_adapt
mcmc::mcmc_sample
param_search::param_wall_check
stoch_grad::stoch_grad_exit
param_search::latin_hypercube
gp_surrogate::gp_update
stoch_grad::stoch_grad_init
gp_surrogate::gp_k_post_same_x
stoch_grad::grad_accent
stoch_grad::nearest_point
stoch_grad::g_t
gradient_estimator::run_gradient_estimator
gradient_estimator::bkwd_diff_f
gradient_estimator::combine_f_grad_k
gradient_estimator::init_diff_f
gp_surrogate::gp_mu_post_grad
gp_surrogate::gp_mu_post_dx
gp_surrogate::gp_post_k_grad
gradient_estimator::get_cholesky
gradient_estimator::m_vector
gradient_estimator::z_vector
gradient_estimator::check_max_is_unique
gradient_estimator::find_max_thread
gp_surrogate::gp_mu_post
basis_functions::reduced_hamiltonian_2_electrons
basis_functions::discrete_laplacian_reduced
basis_functions::reduced_hamiltonian_slater_1s
component_functions::centered_slater_1s_laplacian
basis_functions::wave_function_slater_1s
component_functions::centered_slater_1s
basis_functions::reduced_hamiltonian_sto3g
component_functions::centered_gaussian_laplacian
basis_functions::wave_function_sto3g
component_functions::centered_gaussian
basis_functions::wave_function_2_electrons
component_functions::correlation_function
component_functions::correlation_atom_term
component_functions::correlation_range
component_functions::correlation_subterm
basis_functions::mno_allocate

13.6 bond_driver.f90 File Reference

Driver for running multiple simulations to optimise bond length.

Functions/Subroutines

- program bond_length_driver

  Driver for running multiple simulations to optimise bondlength.

13.6.1 Detailed Description

Driver for running multiple simulations to optimise bond length.

13.6.2 Function/Subroutine Documentation

13.6.2.1 bond_length_driver()

```
program bond_length_driver
```

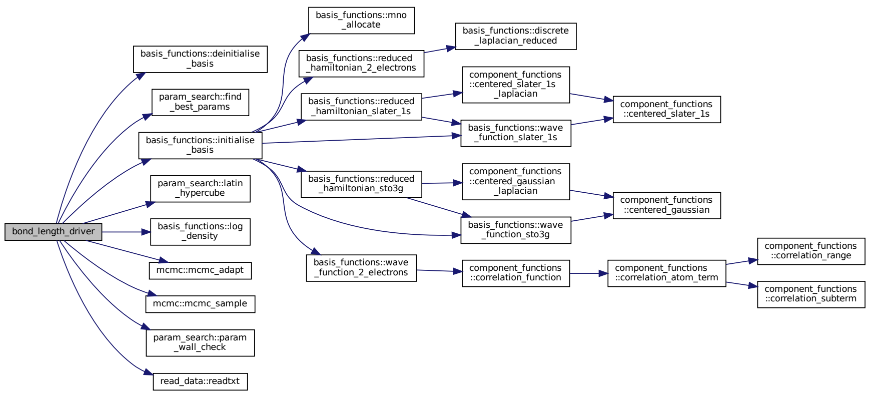
Driver for running multiple simulations to optimise bondlength.

This code computes the ground state energy and corresponding trial wavefunction for a given atom and electron configuration. It saves the resulting wavefunction or electron density in netcdf.

Definition at line 7 of file bond_driver.f90.

References param_search::best_trial, basis_functions::deinitialise_basis(), basis_functions::dof_bounds, shared←↩_constants::dp, param_search::find_best_params(), basis_functions::initialise_basis(), param_search::latin_←↩hypercube(), basis_functions::log_density(), mcmc::mcmc_adapt(), mcmc::mcmc_sample(), basis_functions::n←↩_space_dims, basis_functions::number_dofs, param_search::param_wall_check(), read_data::readtxt(), basis_←↩functions::reduced_hamiltonian, and shared_constants::slater_1s_code.

Here is the call graph for this function:


bond_length_driver
basis_functions::deinitialise_basis
param_search::find_best_params
basis_functions::initialise_basis
param_search::latin_hypercube
basis_functions::log_density
mcmc::mcmc_adapt
mcmc::mcmc_sample
param_search::param_wall_check
read_data::readtxt
basis_functions::mno_allocate
basis_functions::reduced_hamiltonian_2_electrons
basis_functions::reduced_hamiltonian_slater_1s
basis_functions::reduced_hamiltonian_sto3g
basis_functions::wave_function_2_electrons
basis_functions::discrete_laplacian_reduced
component_functions::centered_slater_1s_laplacian
basis_functions::wave_function_slater_1s
component_functions::centered_gaussian_laplacian
basis_functions::wave_function_sto3g
component_functions::correlation_function
component_functions::centered_slater_1s
component_functions::centered_gaussian
component_functions::correlation_atom_term
component_functions::correlation_range
component_functions::correlation_subterm

13.7 calc.f90 File Reference

Function completing calculations of either the wavefunction or electron density (depending on the system) at a set of coordinates [x,y,0] on a grid of equally spaced points on a xy grid.

Modules

- module calculations

  Function completing calculations of either the wavefunction or electron density (depending on the system) at a set of coordinates [x,y,0] on a grid of equally spaced points on a xy grid.

Functions/Subroutines

- real(dp) function, dimension(:), allocatable calculations::calc (dof, points, box, n_ele, n_MCMC_steps)

  calc function returns the wavefunction or electron density evaluated on a 2D xy plane (taking a slice) depending on the number of electrons in the system.

13.7.1 Detailed Description

Function completing calculations of either the wavefunction or electron density (depending on the system) at a set of coordinates [x,y,0] on a grid of equally spaced points on a xy grid.

13.8 component_functions.f90 File Reference

Basic subfunctions used throughout the simulation.

Modules

- module component_functions

  Basic subfunctions This module contains basic functions used in basis_functions.f90.

Functions/Subroutines

- real(dp) function component_functions::centered_gaussian (position, alpha)

  Gaussian distribution centred at the origin.
- real(dp) function component_functions::centered_gaussian_laplacian (position, alpha)

  Analytic Laplacian of Gaussian distribution centred at the origin.
- real(dp) function component_functions::centered_slater_1s (position, zeta)

  Slater-1s distribution centred at the origin.
- real(dp) function component_functions::centered_slater_1s_laplacian (position, zeta)

  Analytic Laplacian of Slater-1s distribution centred at the origin.
- real(dp) function component_functions::correlation_range (r, d)

  Functions that construct the Jastrow interaction function Notation here follows Schmidt and Moskowitz 1990, refered to as SM90.
- real(dp) function component_functions::correlation_subterm (r_12, r_I1, r_I2, m, n, o)

  Jastrow subfuction: basic subterm Subterm of the correlation function, one for each Jastrow dof per atom This is the term in the k sum in Schmidt and Moskowitz 1990.
- real(dp) function component_functions::correlation_atom_term (atom_coord, electron_coords, mno_↩ parameters, c, b, d)

  Jastrow subfuction: atom subterm Correlation term for each atom.
- real(dp) function component_functions::correlation_function (atom_coords, electron_coords, mno_↩ parameters, c, b, d)

  Jastrow correlation fuction Function F from Schmidt and Moskowitz 1990.

13.8.1 Detailed Description

Basic subfunctions used throughout the simulation.

13.9 constants.f90 File Reference

Fortran shared constants definitions.

Modules

- module shared_constants

  Definitions of shared constants used throughout the software.

Variables

- integer, parameter shared_constants::dp = real64
- real(dp), parameter shared_constants::pi = 3.141592653589793238_dp
- real(dp), parameter shared_constants::electronvolt = 27.211386245988_dp
- integer, parameter shared_constants::slater_1s_code = 100
- integer, parameter shared_constants::sto_3g_code = 200

13.9.1 Detailed Description

Fortran shared constants definitions.

13.10 ed_test.f90 File Reference

Driver for testing electron density function.

Functions/Subroutines

- program main_driver

  Driver for testing electron density function.

13.10.1 Detailed Description

Driver for testing electron density function.

13.10.2 Function/Subroutine Documentation

13.10.2.1 main_driver()

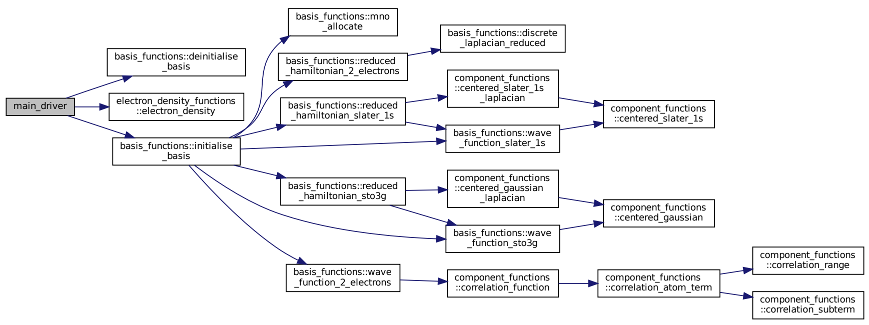
```
program main_driver
```

Driver for testing electron density function.

Definition at line 3 of file ed_test.f90.

References basis_functions::deinitialise_basis(), electron_density_functions::electron_density(), and basis_↩ functions::initialise_basis().

Here is the call graph for this function:


main_driver
basis_functions::deinitialise_basis
electron_density_functions::electron_density
basis_functions::initialise_basis
basis_functions::mno_allocate
basis_functions::reduced_hamiltonian_2_electrons
basis_functions::reduced_hamiltonian_slater_1s
basis_functions::reduced_hamiltonian_sto3g
basis_functions::wave_function_2_electrons
basis_functions::discrete_laplacian_reduced
component_functions::centered_slater_1s_laplacian
basis_functions::wave_function_slater_1s
component_functions::centered_gaussian_laplacian
basis_functions::wave_function_sto3g
component_functions::correlation_function
component_functions::centered_slater_1s
component_functions::centered_gaussian
component_functions::correlation_atom_term
component_functions::correlation_range
component_functions::correlation_subterm

13.11 electron_density.f90 File Reference

Electron density function.

Modules

- module electron_density_functions

  Electron density function.

Functions/Subroutines

- real(dp) function electron_density_functions::electron_density (fixed_position, integral_bounds, dof_↩ coefficients, n_MC_points, seed_in)

  Computes electron density for 2 electron simulations Uses basic Monte Carlo Integration.
- real(dp) function electron_density_functions::wave_function_normalisation (integral_bounds, dof_↩ coefficients, n_MC_points, seed_in)

  Computes the integral of the wavefunction squared To normalise the wavefunction for outputing Uses basic Monte Carlo Integration.

13.11.1 Detailed Description

Electron density function.

13.12 energy_plotting.py File Reference

Script to output the energy against the bond length.

Namespaces

- energy_plotting

  Script outputs the energy against the bond length.

Variables

- energy_plotting.data = np.loadtxt('energies.txt')

  Loading in the data from the energies.txt.
- energy_plotting.bond_length = data[:,0]
- energy_plotting.energy = data[:,1]

13.12.1 Detailed Description

Script to output the energy against the bond length.

13.13 files.txt File Reference

13.14 GP_surrogate.f90 File Reference

Gaussian process surrogate subroutines and functions.

Data Types

- interface gp_surrogate::mean_func_interface
- interface gp_surrogate::mean_func_interface_dx
- interface gp_surrogate::cov_kernal_interface
- interface gp_surrogate::cov_kernal_dx_1_interface
- interface gp_surrogate::cov_kernal_xx_dx_interface
- interface gp_surrogate::gp_k_post

  function for postieror covairance kernal use in format x_1, x_2, x1_dim, x2_dim for cov(x_1,x_2) use in format x, x_dim for cov(x,x)
- interface gp_surrogate::gp_init

  general initialisation function, see gp_init_gausscov

Modules

- module gp_surrogate

  Gaussian process surrogate submodules and functions.

Functions/Subroutines

- subroutine gp_surrogate::gp_init_gausscov (mean_prior, mean_prior_dx, ker_var, ker_length, param_data↩_in, E_data_in, n_data_in, n_dof_in, n_threads_in)

  intialises with a gaussian covariance
- subroutine gp_surrogate::gp_init_arbcov (mean_prior, cov_prior, param_data_in, E_data_in, n_data_in, n↩_dof_in)

  do not use, don't have a function for the dervatives of the kernal/mean
- real(dp) function, dimension(x_dim) gp_surrogate::gp_mu_post (x, x_dim)

  posterior mean
- real(dp) function gp_surrogate::gp_mu_post_dx (x, dim)

  derivative of the posterior mean, wrt dimension dim
- real(dp) function, dimension(x_dim, n_dof) gp_surrogate::gp_mu_post_grad (x, x_dim, only)

  derivative of the posterior mean, needed for stoch_grad
- real(dp) function, dimension(x_dim, x_dim) gp_surrogate::gp_k_post_same_x (x, x_dim)

  specific function for cov(x,x), see gp_k_post
- real(dp) function, dimension(x1_dim, x2_dim) gp_surrogate::gp_k_post_diff_x (x1, x2, x1_dim, x2_dim)

  specific function for cov(x_1,x_2), see gp_k_post
- subroutine gp_surrogate::gp_post_k_grad (X, x_dim, out)

  grad of the posterior covariance, evaluated at X of size x_dim,n_dof, needed for stoch_grad
- subroutine gp_surrogate::gp_update (param_data_in_top, E_data_in_top, n_top, Algo_choice)

  update routine for adding data to gp
- subroutine gp_surrogate::gp_debug_print_state ()

  prints all scalar stae variables, size of all array state variables, and min and max stored energy
- subroutine gp_surrogate::gp_exit ()

  deallocates state variables
- subroutine gp_surrogate::gp_min_stored (min_params, min_E)

  for finding minimum of the energy and associated parameter space point, from stored data
- subroutine gp_surrogate::gp_return_size_data (n_data_out, n_dof_out)

  returns the integers that control the size of the state variables
- subroutine gp_surrogate::gp_return_state_data (kernel_var_out, kernal_inv_length_out, param_data_out, E_data_out, param_pres_out, param_cov_out, data_mean_out)

  for acessing a copy of the state variables, they are returned to the intent(out) parameter with the corresponding name
- subroutine gp_surrogate::gp_restart (n_data_in, n_dof_in, kernel_var_in, kernal_inv_length_in, param_↩data_in, E_data_in, param_pres_in, param_cov_in, data_mean_in, mean_prior, mean_prior_dx, n_threads↩_in)

  sets a state from inputed data, intended for use with restart files.

Variables

- procedure(mean_func_interface_dx), pointer gp_surrogate::mu_prior_dx => null()

13.14.1 Detailed Description

Gaussian process surrogate subroutines and functions.

13.15 GP_surrogate_test_driver.f90 File Reference

Driver for testing the Gaussian process surrogate.

Functions/Subroutines

- program main

  Driver for testing the Gaussain process surrogate.

13.15.1 Detailed Description

Driver for testing the Gaussian process surrogate.

13.15.2 Function/Subroutine Documentation

13.15.2.1 main()

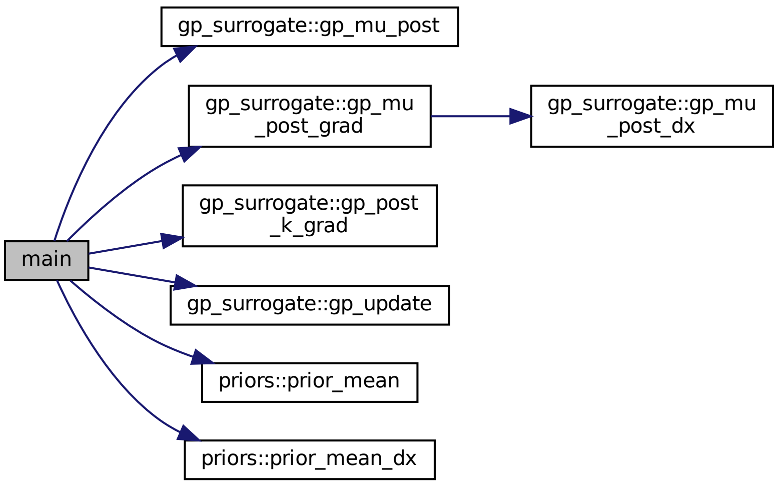
```
program main
```

Driver for testing the Gaussain process surrogate.

Definition at line 3 of file GP_surrogate_test_driver.f90.

References gp_surrogate::gp_mu_post(), gp_surrogate::gp_mu_post_grad(), gp_surrogate::gp_post_k_grad(), gp_surrogate::gp_update(), priors::prior_mean(), and priors::prior_mean_dx().

Here is the call graph for this function:



13.16 gradient_estimator.f90 File Reference

Subroutines for obtaining a gradient estimation through Cholesky decomposition.

Modules

- module gradient_estimator

    Subroutines for obtaining a gradient estimation through Cholesky decomposition.

Functions/Subroutines

- subroutine gradient_estimator::get_cholesky (cholesky_decomp, matrix_in)

    performs Cholesky decomposition without destroying the old array.
- real(dp) function, dimension(:), allocatable gradient_estimator::m_vector (x, previous_best)

    builds the "m_vector" as described in: arXiv:1602.05149v4 [stat.ML] 5 May 2019 contains the difference between the mean and best result in a vector.
- real(dp) function, dimension(:), allocatable gradient_estimator::z_vector (z_in)

    build the "z_vector" as described in: arXiv:1602.05149v4 [stat.ML] 5 May 2019
- integer function, dimension(1) gradient_estimator::find_max_thread (m_vec, z_vec, c_mat)

    finds the location in the vector that corresponds to the maximum of "m + CZ"
- subroutine gradient_estimator::check_max_is_unique (m_vec, z_vec, c_mat, find_max_thread)

    check if the maximum found in the find_max_thread is a unique value.
- subroutine gradient_estimator::init_diff_f (x, z_in, previous_best)

    initialises the workspace F and Cholesky decomposition of the covariance matrix.
- subroutine gradient_estimator::bkwd_diff_f ()

    performs the backward difference differentiation on the workspace F
- subroutine gradient_estimator::combine_f_grad_k (x, n_params)

    combines the gradient contributions from the workspace F and those associated with the m_vector and covariance matrix.
- real(dp) function, dimension(:,:), allocatable gradient_estimator::run_gradient_estimator (x, z_in, n_params, previous_best)

    runs the gradient estimator associated routines from one block.

Variables

- real(dp), dimension(:,:), allocatable gradient_estimator::f_ij
- real(dp), dimension(:,:), allocatable gradient_estimator::l_ij
- real(dp), dimension(:,:), allocatable gradient_estimator::gradient_matrix
- real(dp) gradient_estimator::check_max

13.16.1 Detailed Description

Subroutines for obtaining a gradient estimation through Cholesky decomposition.

13.17 init_params.py File Reference

Python code to obtain user input through GUI or command line.

Namespaces

- init_params

    Python functions and scripts to obtain user input import sympy as sym.

Functions

- def init_params.get_input (parameter='parameter')

    Function to get a numerical value/parameter through command line.
- def init_params.new_param (window, tk_text, tk_entry, tk_input, row, dval='1', text='param', font=font, fs=fontsize)

    Function for creating a row of user entry within the GUI.

Variables

- string init_params.title = 'User Input'
- string init_params.bg_color = 'white'
- string init_params.fg_color = 'grey'
- string init_params.bg_button = 'grey'

    note: Mac OS may not support changing button color
- string init_params.fg_button = 'black'
- string init_params.font = 'Times New Roman'
- int init_params.fontsize = 18
- int init_params.length = 1000
- int init_params.height = 1000
- init_params.window = tk.Tk()
- init_params.background
- string init_params.txt0
- init_params.text0 = tk.Label(window,text=txt0,font=(font,fontsize),wraplength=length-5)

    Initializing/Configuring the top row text.
- init_params.row
- init_params.column
- init_params.columnspan
- init_params.sticky
- list init_params.params
- list init_params.def_values = [1,1,1.5,40,1000000,0,10,100,1,10,12345,1,7,0.01,1.0,1.0,5.0,20]
- list init_params.floats = [2, 13, 14, 15, 16]
- list init_params.integers = [0, 1, 3, 4, 5, 6, 7,8, 9, 10, 11, 12, 17]
- init_params.N_params = len(params)
- string init_params.txt1 = "The user must provide the following inputs: "
- init_params.text1 = tk.Label(window,text=txt1,font=(font,fontsize),wraplength=length-10)
- init_params.dval
- init_params.text
- string init_params.txt2 = "Biased Optimizer Options (not enabled by default):"
- init_params.text2 = tk.Label(window,text=txt2,font=(font,fontsize),wraplength=length-10)
- init_params.pady
- string init_params.txt2_1 = "Enable the biased optimizer?"

    Create a radio button for user to choose between two options.
- init_params.text2_1 = tk.Label(window,text=txt2_1,font=(font,fontsize),wraplength=length-10)
- init_params.bopt = tk.IntVar()
- init_params.variable
- init_params.value
- init_params.indicatoron
- init_params.padx
- init_params.entry5 = bopt
- int init_params.i = 6
- string init_params.txt3 = "Optional user inputs (default options are shown):"
- init_params.text3 = tk.Label(window,text=txt3,font=(font,fontsize),wraplength=length-10)

- init_params.basis = tk.IntVar()
- init_params.entry7 = basis
- string init_params.txt4 = "Visualization options:"
- init_params.text4 = tk.Label(window,text=txt4,font=(font,fontsize),wraplength=length-10)
- init_params.close
- init_params.relx
- init_params.rely
- init_params.anchor
- int init_params.p1 = 1

  Checks to see if user input is proper and sets everything to either an integer or float.
- int init_params.p2 = 0
- string init_params.filename = 'init_params.txt'
- init_params.file = open(filename,'w')

13.17.1 Detailed Description

Python code to obtain user input through GUI or command line.

13.18 init_params.txt File Reference

13.19 latin_driver.f90 File Reference

Simulation driver for latin hypercube sampling.

Functions/Subroutines

- program main_driver

  Main driver for simulation using the latin_hypercube search.

13.19.1 Detailed Description

Simulation driver for latin hypercube sampling.

13.19.2 Function/Subroutine Documentation

13.19.2.1 main_driver()

```
program main_driver
```

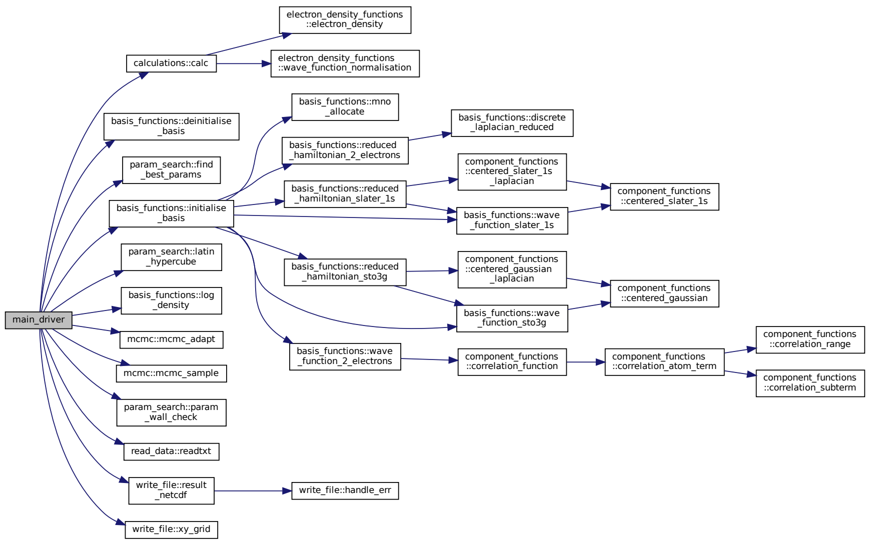
Main driver for simulation using the latin_hypercube search.

This code computes the ground state energy and corresponding trial wavefunction for a given atom and electron configuration. It saves the resulting wavefunction or electron density in netcdf.

Definition at line 7 of file latin_driver.f90.

References param_search::best_params, param_search::best_trial, calculations::calc(), basis_functions↩ ::deinitialise_basis(), basis_functions::dof_bounds, shared_constants::dp, shared_constants::electronvolt, param↩ _search::find_best_params(), basis_functions::initialise_basis(), param_search::latin_hypercube(), basis_↩ functions::log_density(), mcmc::mcmc_adapt(), mcmc::mcmc_sample(), basis_functions::n_space_dims, basis↩ _functions::number_dofs, param_search::param_wall_check(), read_data::readtxt(), basis_functions::reduced_↩ hamiltonian, write_file::result_netcdf(), shared_constants::slater_1s_code, and write_file::xy_grid().

Here is the call graph for this function:


electron_density_functions ::electron_density
calculations::calc
electron_density_functions ::wave_function_normalisation
basis_functions::mno _allocate
basis_functions::deinitialise _basis
basis_functions::discrete _laplacian_reduced
basis_functions::reduced _hamiltonian_2_electrons
param_search::find _best_params
component_functions ::centered_slater_1s _laplacian
basis_functions::reduced _hamiltonian_slater_1s
component_functions ::centered_slater_1s
basis_functions::initialise _basis
basis_functions::wave _function_slater_1s
param_search::latin _hypercube
basis_functions::reduced _hamiltonian_sto3g
component_functions ::centered_gaussian _laplacian
basis_functions::log _density
component_functions ::centered_gaussian
main_driver
basis_functions::wave _function_sto3g
mcmc::mcmc_adapt
component_functions ::correlation_range
basis_functions::wave _function_2_electrons
component_functions ::correlation_function
component_functions ::correlation_atom_term
component_functions ::correlation_subterm
mcmc::mcmc_sample
param_search::param _wall_check
read_data::readtxt
write_file::result _netcdf
write_file::handle_err
write_file::xy_grid

13.20 main_driver.f90 File Reference

Main driver script for simulation.

Functions/Subroutines

- program main_driver

  Main driver for simulation.

13.20.1 Detailed Description

Main driver script for simulation.

13.20.2 Function/Subroutine Documentation

13.20.2.1 main_driver()

```
program main_driver
```

Main driver for simulation.

More description to be added

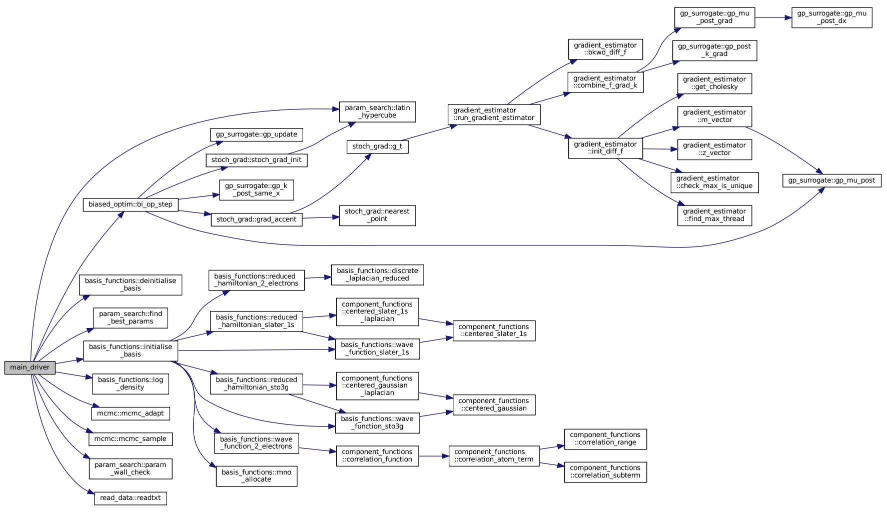
Parameters

n_electrons←↩ _in	Number of electrons

Definition at line 6 of file main_driver.f90.

References param_search::best_params, param_search::best_trial, biased_optim::bi_op_step(), basis_functions←↩ ::deinitialise_basis(), basis_functions::dof_bounds, param_search::find_best_params(), basis_functions::initialise←↩ _basis(), param_search::latin_hypercube(), basis_functions::log_density(), mcmc::mcmc_adapt(), mcmc::mcmc←↩ _sample(), basis_functions::n_space_dims, basis_functions::number_dofs, param_search::param_wall_check(), read_data::readtxt(), and basis_functions::reduced_hamiltonian.

Here is the call graph for this function:


main_driver
biased_optim::bi_op_step
gp_surrogate::gp_update
stoch_grad::stoch_grad_init
gp_surrogate::gp_k _post_same_x
stoch_grad::grad_accent
param_search::latin _hypercube
stoch_grad::g_t
stoch_grad::nearest _point
gradient_estimator ::run_gradient_estimator
gradient_estimator ::bkwd_diff_f
gradient_estimator ::combine_f_grad_k
gradient_estimator ::init_diff_f
gp_surrogate::gp_mu _post_grad
gp_surrogate::gp_mu _post_dx
gp_surrogate::gp_post _k_grad
gradient_estimator ::get_cholesky
gradient_estimator ::m_vector
gradient_estimator ::z_vector
gradient_estimator ::check_max_is_unique
gradient_estimator ::find_max_thread
gp_surrogate::gp_mu_post
basis_functions::deinitialise _basis
param_search::find _best_params
basis_functions::initialise _basis
basis_functions::log _density
mcmc::mcmc_adapt
mcmc::mcmc_sample
param_search::param _wall_check
read_data::readtxt
basis_functions::reduced _hamiltonian_2_electrons
basis_functions::discrete _laplacian_reduced
basis_functions::reduced _hamiltonian_slater_1s
component_functions ::centered_slater_1s _laplacian
basis_functions::wave _function_slater_1s
component_functions ::centered_slater_1s
basis_functions::reduced _hamiltonian_sto3g
component_functions ::centered_gaussian _laplacian
basis_functions::wave _function_sto3g
component_functions ::centered_gaussian
basis_functions::wave _function_2_electrons
component_functions ::correlation_function
component_functions ::correlation_atom_term
component_functions ::correlation_range
component_functions ::correlation_subterm
basis_functions::mno _allocate

13.21 mainpage.txt File Reference

13.22 MCMC.f90 File Reference

Functions and subroutines for implementing Markov chain Monte Carlo.

Data Types

- interface mcmc::log_rho_interface

Modules

- module mcmc

  Functions and subroutines for implementing Markov chain Monte Carlo.

Functions/Subroutines

- subroutine mcmc::mcmc_sample (samples, log_rho, x_0, n_steps, n_burned, thinning_interval, s, e_code, dof_coefficients, density_dimension, average_accept, seed)

  The main routine for MCMC, generates (n_steps-n_burned)/thinning_interval samples from a distribution rho.
- subroutine mcmc::mcmc_adapt (s_out, log_rho, x_0, n_steps, s_0, e_code, s_max, s_min, memory, adapt↩ _interval, dof_coefficients, density_dimension, seed)

  Runs a version of mcmc_sample but every adapt_interval steps adjusts s, based on an exponetial average of the accpetance rate.

13.22.1 Detailed Description

Functions and subroutines for implementing Markov chain Monte Carlo.

13.23 netcdf_file.f90 File Reference

Fortran subroutines for writing results to file.

Modules

- module write_file

  Contains all the subroutines related to writing results to file.

Functions/Subroutines

- subroutine write_file::result_netcdf (dof, ele, num_ele, num_nuc)

  Main results writing to NetCDF file.
- subroutine write_file::energies_netcdf (energies, bondlen)

  Output results for bond lengths and corresponding energy used for plotting.
- subroutine write_file::xy_grid (points, box_size)

  Writes the xyz coordinates used in the calculation of either the wavefunction or electron density to a text file names xyz.txt.
- subroutine write_file::write_restart_file (gp_n_data, gp_n_dof, n_cycles, no_samples, current_best_E, constant_mean_value, gamma, kernel_var, kernal_inv_length, E_data, data_mean, param_pres, param_cov, param_data)

  Creates a NetCDF file which contains information needed for the restart of the Bi_Op_step function.
- subroutine write_file::handle_err (ierr)

  Handles the NetCDF errors and print statements.

13.23.1 Detailed Description

Fortran subroutines for writing results to file.

13.24 param_search.f90 File Reference

Functions/subroutines associated with building/initialisation the parameter search space and finding the best parameters from a given MCMC run.

Modules

- module param_search

  Functions/subroutines associated with building/initialisation the parameter search space and finding the best paramters from a given MCMC run.

Functions/Subroutines

- subroutine param_search::random_search_grid (trials, param_bounds, n_trials, seed_in)

  random_search_grid returns MxN grid of test points for M free parameters and N trials distributed
- subroutine param_search::latin_hypercube (trials, param_bounds, n_trials, seed_in)

  latin_hypercube returns MxN grid of test points for M free parameters and N trials on a latin hypercube for given bounds.
- subroutine param_search::find_best_params (trial_energies, trials)

  find_best_params returns the 1D array that contain the current best set of parameters found from a MCMC run.
- subroutine param_search::param_wall_check (param_bounds)

  param_wall_check evaluates if the best parameter set is close to the parameter bounds used to generate the test points.

Variables

- real(dp), dimension(:), allocatable, protected param_search::best_params
- integer, dimension(1), protected param_search::best_trial

13.24.1 Detailed Description

Functions/subroutines associated with building/initialisation the parameter search space and finding the best parameters from a given MCMC run.

13.25 plotting.py File Reference

Python plotting scripts which output the contour plots for the wave function or electron probability density.

Namespaces

- plotting

  Main plotting script which outputs the contour plots for the wavefunction or electron probability density.

Variables

- plotting.coords = np.loadtxt('xyz.txt')

  Uploading the xy plane coordinates from the xyz.txt file generated from the xy_grid subroutine in netcdf_file.f90.
- plotting.data = nc.Dataset('results.nc4', mode='r', format='NETCDF4')

  Uploading the main results NetCDF file named results.nc4 generated from the result_netcdf routine.
- plotting.num_ele = data.variables['Num_of_Electrons'][:]

  Prints a statement if it has been uploaded successfully.
- plotting.num_nuc = data.variables['Num_of_Nuclei'][:]
- plotting.x = coords[:,0]

  Extracts the x and y coordinates from the xyz.txt.
- plotting.y = coords[:,1]
- plotting.wavefunction = data.variables['Electron_Density'][:]

  This does the contour plot for the 1 electron case.
- plotting.dof = data.variables['Optimal_DOF'][:]
- plotting.ele_den = data.variables['Electron_Density'][:]

  Creates the contour plot of the wavefunction squared.

13.25.1 Detailed Description

Python plotting scripts which output the contour plots for the wave function or electron probability density.

13.26 priors.f90 File Reference

Functions for obtaining the prior mean and its derivative.

Modules

- module priors

  Functions for obtaining the prior mean and its derivative.

Functions/Subroutines

- real(dp) function priors::prior_mean (x)

  Gives the prior mean.
- real(dp) function priors::prior_mean_dx (x, dim)

  Derivative of the prior mean.

13.26.1 Detailed Description

Functions for obtaining the prior mean and its derivative.

13.27 read_data.f90 File Reference

Subroutines to read files of user input into fortran.

Modules

- module read_data

    Subroutines to read files and transfer user inputs into fortran.

Functions/Subroutines

- subroutine read_data::readtxt (filename, n_electrons_in, n_atoms_in, bond_length, n_trials, n_MCMC↩_steps, n_omp_threads, use_biased_optimiser, n_optimiser_steps, basis_type_in, proton_number_int_in, search_seed, n_basis_functions_per_atom_in, n_Jastrow_in, fd_length_in, Jastrow_b_length_in, Jastrow↩_d_length_in, plot_distance, plot_points)

    Read in data from a text file.

13.27.1 Detailed Description

Subroutines to read files of user input into fortran.

13.28 stoch_grad.f90 File Reference

Modules for optimization.

Modules

- module stoch_grad

    Optimization modules.

Functions/Subroutines

- subroutine stoch_grad::stoch_grad_init (N, best_from_last_run_in, seed)

    Initalises module variables and sets up starts points for the restarts of the gradient assent.
- subroutine stoch_grad::stoch_grad_exit ()

    deallocates state variables
- subroutine stoch_grad::grad_accent (X_0, sequence_length, no_samples, gamma, out)

    Performs the gradient assent for sequence length X_0 is the intial point to start the assent at sequence_length is how long the sequence to average over is No_samlples is the number of samples taken in the gradient estimator, gamma is parameter to tune for how much to move the next point long the estimated gradient out is the output and used to compute the sum on the go.
- real(dp) function, dimension(x_shape(1), x_shape(2)) stoch_grad::g_t (X, no_samples, X_shape)

    Return an estimate for the gradient which is an average over no_samples.
- real(dp) function, dimension(:,:), allocatable stoch_grad::nearest_point (bounds, X)

    Returns a new point back to the search space if it leaves.

Variables

- integer stoch_grad::n_restarts
- real(dp), dimension(:,:,:), allocatable stoch_grad::n_points

13.28.1 Detailed Description

Modules for optimization.

13.29 tutorial.txt File Reference

13.30 vmc.txt File Reference