EMPIRE DA

v1.9.1

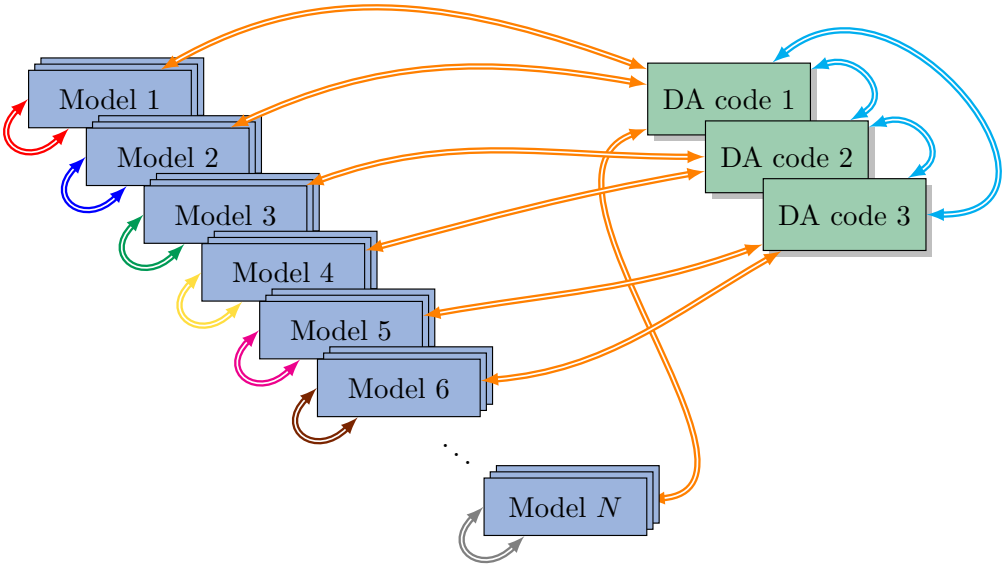
Generated by Doxygen 1.8.8

Tue Aug 16 2016 16:56:38

Contents

Chapter 1

EMPIRE Data Assimilation Documentation



Author

Philip A. Browne p.browne@reading.ac.uk

Date

Time-stamp: <2016-08-16 16:27:07 pbrowne>

Contributors

- Mengbin Zhu - zhumengbin @ gmail.com
- David Scott - d . scott @ ed . ac . uk - Funded by an eCSE project

1.1 EMPIRE Methods

For a list of methods implemented in EMPIRE, please click here: methods

1.2 Downloading

For standalone downloads of the code, please visit https://bitbucket.org/pbrowne/empire-data-assimilation/ click on "Tags" and download the version of your choosing.

For the most up-to-date versions of the code, they are hosted on www.bitbucket.org and can be obtained with the following commands:

```
1 git clone https://pbrowne@bitbucket.org/pbrowne/empire-data-assimilation.git
```

To upgrade to the latest versions of the codes, use the following command:

```
1 git pull https://pbrowne@bitbucket.org/pbrowne/empire-data-assimilation.git
```

Copyright

1.3 Compiling

1.3.1 Compilation of the source code

The Makefile must be edited for the specific compiler setup. In the main directory you will find the file `Makefile.in`.

Edit the variables as follows:

- `FC` The fortran compiler

This has been tested with gfortran 4.8.2, crayftn 8.2.6 and ifort 14.0.1.106

- `FCOPTS` The options for the fortran compiler
- `LIB_LIST` The libraries to be called. Note this must include BLAS and LAPACK
- `MODFLAG` The flag to specify where module files should be placed by the fortran complier. Examples are
  - `gfortran:` -J
  - `ifort:` -module
  - `crayftn:` -em -J
  - `pgfortran:` -module

To compile the source code, simply then type the command

```
1 make
```

If successful, the following executables are created in the bin/ folder:

- empire
- alltests
- test_hqhtr
- test_q
- test_r

To remove the object and executable files if compilation fails for some reason, run the following:

```
1 make clean
```

1.3.2 Compilation of the documentation

Documentation of the code is automatically generated using Doxygen, dot and pdflatex.

All of these packages must be installed for the following to work.

```
1 make docs
```

This will make an html webpage for the code, the mainpage for which is located in doc/html/index.html.

A latex version of the documentation will be built to the file doc/latex/refman.pdf.

To simply make the html version of the documentation (if pdflatex is not available) then use the command

```
1 make doc_html
```

1.4 Customising for specific models

This is where the science and all the effort should happen!!

First, the communication version that one wishes to use should be selected. This is done by setting the parameter comm_version in comm_version.f90 . This will define how the state vector is passed between empire and the model and how it is distributed over MPI processes. See Communication Methods for more details.

The file model_specific.f90 should be editted for the specific model which you wish to use. This contains a number of subroutines which need to be adapted for the model and the observation network. We list these subsequently.

- configure_model This is called early in the code and can be used to read in any data from files before subsequently using them in the below operations.
- reconfigure_model This is called after each observation timestep. If the observation dimension changes it should be updated here, along with the number of model timesteps until the next observation
- h This is the observation operator
- ht This is the transpose of the observation operator
- r This is the observation error covariance matrix R
- rhalf This is the square root of the observation error covariance matrix $R^{\frac{1}{2}}$
- solve_r This is a linear solve with the observation error covariance matrix, i.e. given b, find x such that $Rx = b$ or indeed, $x = R^{-1}b$
- solve_rhalf This is a linear solve with the square root of the observation error covariance matrix, i.e. given b, find x such that $R^{\frac{1}{2}}x = b$ or indeed, $x = R^{-\frac{1}{2}}b$
- q This is the model error covariance matrix Q
- qhalf This is the square root model error covariance matrix $Q^{\frac{1}{2}}$
- solve_hqht_plus_r This is a linear solve with the matrix $(HQH^T + R)$
- dist_st_ob This specifies the distance between a an element of the state vector and an element of the observation vector
- bhalf This is the square root of the background error covariance matrix $B^{\frac{1}{2}}$
- get_observation_data This subroutine must return the observation data at, or subsequently to, the given timestep. This routine only needs to be editted if you wish to use your own observations. It is set up to work automatically with pseudo-observations for running twin experiments.

Not all of these subroutines will be required for each filtering method you wish to use, so it may be advantageous to only implement the necessary ones.

1.5 Testing

You can test your user supplied routines by running the test codes found in the folder bin/.

These are by no means full-proof ways of ensuring that you have implemented things correctly, but should at least check what you have done for logical consistency.

For example, they will test if $R^{-1}Ry = y$, and if $Q^{\frac{1}{2}}Q^{\frac{1}{2}}x = Qx$ for various different vectors x, y.

1.6 Linking to your model using EMPIRE

Full instructions on how to put the EMPIRE MPI commands into a new model can be found at www.met.↩ reading.ac.uk/~darc/empire.

1.7 Running

For example, to run **N_MDL** copies of the model with **N_DA** copies of empire, then the following are possible:

```
1 mpirun -np N_MDL model_executable : -np N_DA empire
```

```
1 aprun -n N_MDL -N N_MDL model_executable : -n N_DA -N N_DA empire
```

The empire executable is controlled by the namelist data file empire.nml. As such, this file should be put in the directory where empire is executed.

1.8 Examples

In the directory `examples` there is currently one example of how to use EMPIRE, specifically with the Lorenz 1996 model. In the directory you will find an example model_specific.f90 file setup for that model, along with a file `instructions.txt` which will lead you step by step through how to run a twin experiment.

1.9 Bug Reports and Functionality Requests

While the code is not too large, you may email me the issue or request here.

However there is a webpage set up for this:

https://bitbucket.org/pbrowne/empire-data-assimilation/issues

Chapter 2

How to Cite EMPIRE

EMPIRE itself

For all applications that use these codes, please cite the following paper:

PA Browne, S Wilson (2015)

A simple method for integrating a complex model into an ensemble data assimilation system using MPI

http://dx.doi.org/10.1016/j.envsoft.2015.02.003

Use of different methods within EMPIRE

Equivalent weights particle filter

Van Leeuwen (2010)

Nonlinear data assimilation in geosciences: an extremely efficient particle filter

http://doi.wiley.com/10.1002/qj.699

Sequential importance resampling

Gordon, Salmond and Smith (1993)

Novel approach to nonlinear/non-Gaussian Bayesian state estimation

http://dx.doi.org/10.1049/ip-f-2.1993.0015

Localised Ensemble Transform Kalman Filter

Hunt, Kostelich and Szunyogh (2007)

Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter

http://dx.doi.org/10.1016/j.physd.2006.11.008

4DEnVar

Liu, Xian and Wang (2008)

An Ensemble-Based Four-Dimensional Variational Data Assimilation Scheme. Part I: Technical Formulation and Preliminary Test

http://dx.doi.org/10.1175/2008MWR2312.1

Use of different external codes within EMPIRE

CG+

Gilbert and Nocedal (1992)

Global Convergence Properties of Conjugate Gradient Methods for Optimization

http://dx.doi.org/10.1137/0802003

Software available here: http://users.iems.northwestern.edu/~nocedal/CG+.html

L-BFGS-B

Byrd, Lu and Nocedal (1995)

A Limited Memory Algorithm for Bound Constrained Optimization

http://dx.doi.org/10.1137/0916069

and/or

Zhu, Byrd and Nocedal (1997)

L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization

http://dx.doi.org/10.1145/279232.279236

Software available here: http://users.iems.northwestern.edu/~nocedal/lbfgsb.html

Chapter 3

Communication Methods

EMPIRE has currently 5 different standards for communicating with models. Each method has various advantages and disadvantages. This is changed by modifying the parameter comm_version in comm_version.f90.

Here we list some particularities of each method, before summarising in the table below.

3.1 EMPIRE communication version 1

Here, the state vector is gathered onto a single model process, before being sent to EMPIRE via a single mpi_send call. EMPIRE reverses the process via a single mpi_send of the entire state vector to the single model process, where it scatters the state vector to the rest of the model processes.

Advantages	Disadvantages
EMPIRE is launched on a single process for each ensemble member, making the coding here very simple	EMPIRE is launched on a single process, limiting the size of the state vector
The state vector is explicitly organised	Speed may be compromised by the 2-stage mpi process
	Memory is required on the master process of the model to store the whole state vector
	Knowledge of the model's mpi structure is needed to perform the gather and scatter

3.2 EMPIRE communication version 2

Here, the state vector is gathered directly onto the EMPIRE process. The reverse is that EMPIRE scatters the state vector directly back to the model processes.

Advantages	Disadvantages
EMPIRE is launched on a single process for each ensemble member, making the coding here very simple	EMPIRE is launched on a single process, limiting the size of the state vector
Speed is reasonable due to the 1-stage mpi process	The state vector is organised by the mpi_gather process
No extra memory in the model processes are required	

3.3 EMPIRE communication version 3

Here, EMPIRE has a similar parallel structure as the model. Each model process sends only its local part of the whole state vector to the corresponding EMPIRE process via an mpi_send call. EMPIRE reverses the process with

a direct mpi_send of its own.

Advantages	Disadvantages
EMPIRE is launched on multiple mpi processes per ensemble member, thus limiting the size of the model used by the size of the whole HPC machine, not the memory on one node	EMPIRE is launched on multiple mpi processes per ensemble member, hence the coding of model specific operators becomes much more complicated (if they are not simply diagonal)
Speed is very high thanks to local communications	Not fully tested [please get in touch if you want to help here with your models :)]
No extra memory is required in the model processes	

3.4 Comparison of EMPIRE communication versions

Feature	Version 1	Version 2	Version 3
Number of mpi processes per ensemble member	1	1	Same as model
Communication method: model to EMPIRE	Model gather, mpi_send	mpi_gather	mpi_send
Communication method: EMPIRE to model	mpi_send, model scatter	mpi_scatter	mpi_send
Efficiency (cpu-time)	fast	faster	fastest
Model memory requirements	high	low	low
EMPIRE memory requirements	high	high	low
Ease of model specific implementations	easiest	easy	hard

3.5 EMPIRE communication version 4

Here the model is a directly a subroutine of EMPIRE. This may be useful for toy models.

The disadvantage of this is that, for models of any complexity, it will be next to impossible to turn them into a subroutine. However it avoids using MPI communication so should be as efficient as possible.

3.6 EMPIRE communication version 5

Here the communication is similar to that on version 2, however each instance of the model can run multiple ensemble members. This is designed specifically for use with TOMCAT

Chapter 4

EMPIRE communicators

On this page we descibe the MPI communicators used within the EMPIRE codes to communicate within the data assimilation processes.

4.1 Versions 1, 2, 4 and 5

For comms versions 1 and 2, there is only 1 communicator for this: pf_mpi_comm

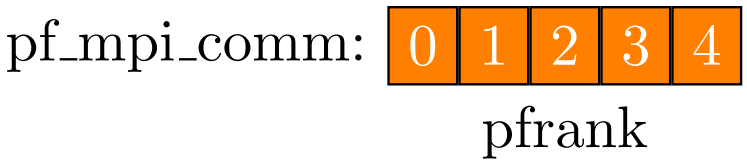


Figure 4.1: pf_mpi_comm

- pf_mpi_comm is undefined, but its size is set to 1
- pf_ens_comm is set to pf_mpi_comm

4.2 Version 3

For comms version 3, there are 3 communicators:

- pf_mpi_comm: All of the DA MPI processes
- pf_ens_comm: The DA MPI processes associated with the ensemble at that point
- pf_member_comm: The DA MPI processes associated with a single ensemble member

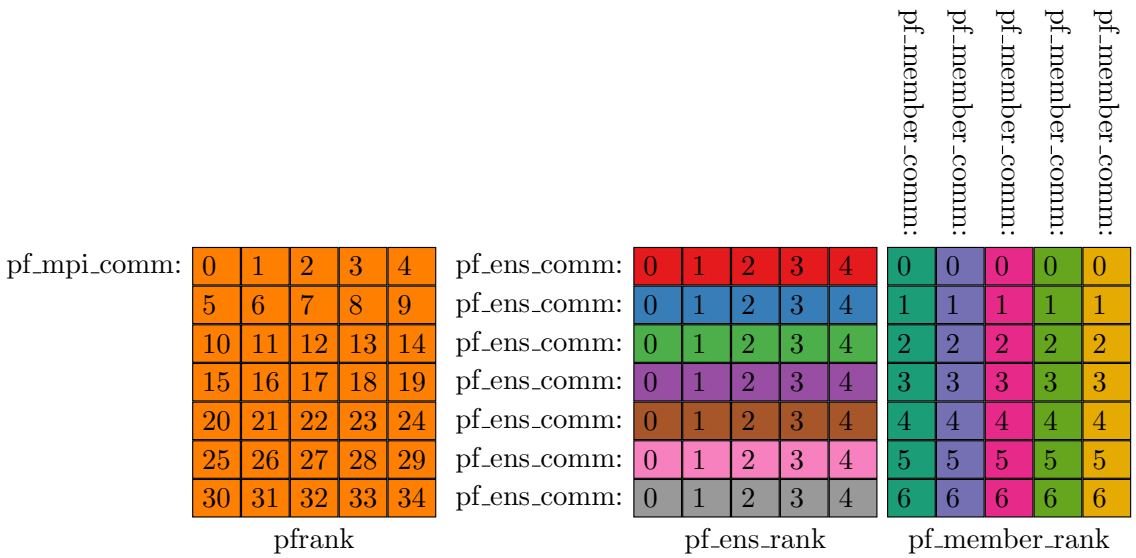


Figure 4.2: version 3 communicators

Chapter 5

Assimilation Methods

5.1 Filters

The filters implemented in EMPIRE can be divided into two categories, particle filters and Ensemble Kalman filters

5.1.1 Particle filters

5.1.1.1 SIR Filter (Sequential Importance Resampling)

See file sir_filter
Gordon, Salmond and Smith (1993).
Model specific operations required:

- qhalf
- h
- solve_r
  The SIR filter has no parameters to be chosen.
  To select the SIR filter, in empire.nml set the following variables:
  - filter = 'SI'

5.1.1.2 Equivalent Weights Particle Filter

See files proposal_filter equivalent_weights_filter
Van Leeuwen (2010).
Model specific operations required:

- qhalf
- q
- h
- ht
- solve_r
- solve_hqht_plus_r The Equivalent Weights particle filter has a number of free parameters to be chosen.

  - nudgefac
  - nfac
  - ufac
  - keep
    To select the EWPF, in empire.nml set the following variables:
  - filter = 'EW'

5.1.1.3 The Zhu and Van Leeuwen Equivalent Weights Particle Filter

See files proposal_filter equivalent_weights_filter_zhu
Model specific operations required:

- qhalf
- q
- h
- ht
- solve_r
- solve_rhalf
- solve_hqht_plus_r The Zhu Equivalent Weights particle filter has a number of free parameters to be chosen.

  - nudgefac
    To select the EZPF, in empire.nml set the following variables:
  - filter = 'EZ'

5.1.2 Ensemble Kalman filters

5.1.2.1 LETKF (The Localised Ensemble Transform Kalman Filter)

See file letkf_analysis
Hunt, Kostelich and Szunyogh (2007).
Model specific operations required:

- h
- solve_rhalf
- dist_st_ob
  The LETKF has a number of free parameters to be chosen.

  - rho
  - len
    To select the LETKF, in empire.nml set the following variables:
  - filter = 'LE' or 'LD' with LE including model error but LD being deterministic

5.2 Smoothers

Coming at some point in the future: LETKS (Please contact us if you want us to develop this sooner rather than later)

5.3 Variational Methods

5.3.1 3DVar

See files threedvar_fcn
Model specific operations required:

- h
- ht
- solve_b
- solve_r

3DVar can be used as a filter in a sequential run. Each particle uses its current forecast as a background guess for an independent 3DVar minimization.

5.3.2 4dEnVar

See files fourdenvar fourdenvar_fcn
Liu, Xian and Wang (2008).
Model specific operations required:

- h
- solve_r

Currently there is the basic functionality to do 4dEnVar so long as EMPIRE-VADER is used for reverse communication. This is work in progress.

Todo Add some stuff about how to use this.

Chapter 6

Other EMPIRE features

6.1 Generating artificial observations

EMPIRE can generate artificial observations easily and quickly.

Model specific operations required:

- h
- rhalf
- qhalf

In empire.nml set the following variables:

- gen_data = .true.

The system then should be run with a single ensemble member and a single EMPIRE process, i.e.

```
1 mpirun -np 1 model : -np 1 empire
```

6.2 Observations

To use real observations (i.e. those not generated automatically in twin experiment mode) the user must change the subroutine get_observation_data in model_specific.f90.

When called, get_observation_data must return the vector of observations y that corresponds to the observation on, subsequently to, the current timestep.

6.3 Running a deterministic ensemble

EMPIRE can simply integrate forward in time an ensemble of models.

In empire.nml set the following variables:

- filter = 'DE'

6.4 Running a stochastic ensemble

EMPIRE can integrate forward in time an ensemble of models whilst adding stochastic forcing.

Model specific operations required:

- qhalf

In empire.nml set the following variables:

- filter = 'SE'

6.5 Redirecting the output from EMPIRE

This feature can be used to suppress output from EMPIRE STDOUT.

See open_emp_o for more information.

6.6 Outputting ensemble member weights

This is controlled by output_weights in empire.nml . By default the weights will not be output. If set to true, this will create a number of files named "ensemble_weights_??" where ?? will refer to the rank of the empire process on pf_mpi_comm. Within that file, the timestep, particle number and the negative log of the weight will be output. Note that these weights may not be normalised.

6.7 Outputting rank histograms

This is controlled by use_talagrand in empire.nml and for more information see load_histogram_data .

6.8 Outputting trajectories of model variables

This is controlled by use_traj in empire.nml and for more information see setup_traj .

6.9 Outputting mean of the ensemble

EMPIRE has the ability to output the mean of the ensemble in each dimension. For each dimension of the state vector j, the ensemble mean $\bar{x}_j$ is defined as

$\bar{x}_j := \frac{1}{N_e} \sum_{i=1}^{N_e} x_{i,j}$ where $x_{i,j}$ is the jth component of ensemble member i and N_e is the number of ensemble members. To use this feature, set use_mean = true in empire.nml.

6.10 Outputting covariances of the ensemble

EMPIRE has the ability to output the ensemble covariance matrices throughout the run. This is controlled by the optional namelist &mat_pf in empire.nml. For more information see matrix_pf::matrix_pf_data. Note however, that this will output a large matrix – if the state dimension of the model is large, this is likely not a good thing to compute! This feature is not available with empire version 3 communications.

6.11 Outputting variances of the ensemble

EMPIRE has the ability to output the variance in the ensemble in each dimension. For each dimension of the state vector j, the ensemble variance σ_j^2 is defined as

$\sigma_j^2 := \frac{1}{N_e-1}\sum_{i=1}^{N_e}(x_{i,j}-\bar{x}_j)^2$ Note that this is the sample variance. To use this feature, set use_variance = true in empire.nml.

6.12 Outputting Root Mean Squared Errors

In a twin experiment, where EMPIRE has generated a "truth", EMPIRE can output the following: $\sqrt{(\frac{1}{N_x}\sum_{i=1}^{N_x}(\bar{x}_i-x_i^t)^2)}$ where N_x is the state dimension (state_dim), $\bar{x}$ is the ensemble mean, x^t is the truth, and i is an index running over each component of the state vector.

Note that in the case where the model has different variables, that are on different scales, this is probably not a good measure. For example, if one component of the state vector is measured in units of "apples per pie" and another is measured in "oranges per country per decade", this measure of RMSE will combine the two. Hence the latter should have much larger scale than the former, so this RMSE measure will be dominated by the errors in the components with greater variability. To use this feature, set use_spatial_rmse = true in empire.nml.

There is now the option to compute RMSE fields using the formula: $f_j(t) = \sqrt{\frac{1}{N_e}\sum_{i=1}^{N_e}(x_{i,j}(t)-x_j^t(t))^2}$ where $f_j(t)$ is the jth component of the state at time t, $x_j^t(t)$ is the jth component of the truth at time t, $x_{i,j}(t)$ is the jth compnonent of ensemble member i at time t and N_e is the number of ensemble members.

This is controlled by the option use_ens_emse in empire.nml .

Chapter 7

Tutorials

Here we have a list of tutorials for using EMPIRE. Hopefully this list will be expanded in the future. Please contribute to these pages to help others!

- Lorenz 96 Tutorial

7.1 Lorenz 96 Tutorial

Author: PA Browne. Time-stamp: <2016-01-16 00:54:55 pbrowne>

7.1.1 Description of the model

$$\frac{\mathrm{d}x_k}{\mathrm{d}t} = -x_{k-2}x_{k-1} + x_{k-1}x_{x+1} - x_k + F$$

Todo Write some actual description of this model

7.1.2 Connecting the model to EMPIRE using MPI.

Todo Write some stuff about this. maybe a separate page.

Fortunately, there is a model already within EMPIRE that can do all this. Build it with the command

```
1 make lorenz96
```

Now you can check that a model executable was created. It will be found in the binaries folder bin/

```
1 ls -l bin/lorenz96
```

7.1.3 Specification of subroutines in model_specific.f90

Before any data assimilation can be done with the model, we must specify operators such as the observation operator H, its error covariance R, and a few others. Below, we list how we shall set these up in the file model_↩ specific.f90, which is located in the top directory of EMPIRE.

7.1.3.1 The observations

7.1.3.1.1 Defining observation operators:

We shall have an observation network that does not change with time. We are going to observe every other grid point directly.

That is, $y = H(x) = \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ \vdots \\ x_{N-1} \end{bmatrix}$.

Note that here we are using Fortran indexing (starting from 1). For simplicity, we will assume that N is odd. Hence the observation operator should look as follows.

H is therefore a matrix in $\mathbb{R}^{\frac{N}{2} \times N}$ such that

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

We never form this matrix, instead we simply implement H (and its transpose, HT) by selecting (inserting) the appropriate values from (to) the state vector. Note that we have to do this for an arbitrary number of state vectors.

```
subroutine h(obsDim,nrhs,x,hx,t)
  use sizes
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: obsdim
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(state_dim,nrhs), intent(in) :: x
  real(kind=rk), dimension(obsDim,nrhs), intent(out) :: hx
  integer, intent(in) :: t
  hx(:,:) = x(1:state_dim:2,:)
end subroutine H
```

Similarly, for the transpose of this observation operator, we can write this as follows:

```
subroutine ht(obsDim,nrhs,y,x,t)
  use sizes
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: obsdim
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(obsDim,nrhs), intent(in) :: y
  real(kind=rk), dimension(state_dim,nrhs), intent(out) :: x
  integer, intent(in) :: t
  x = 0.0_rk
  x(1:state_dim:2,:) = y(:,:)
end subroutine HT
```

7.1.3.1.2 Defining observation error covariances:

Let us assume that we have homogeneous, uncorrelated observation errors such that $R = \sigma^2 I$. For this example, $\sigma^2 = 0.1$. Then we can code multiplication by R in the following way:

```
subroutine r(obsDim,nrhs,y,Ry,t)
  use rdata
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: obsdim
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(obsDim,nrhs), intent(in) :: y
  real(kind=rk), dimension(obsDim,nrhs), intent(out) :: ry
  integer, intent(in) :: t
  ry = y*0.1d0
end subroutine R
```

Similarly, application of $R^{\frac{1}{2}}$ can be done as:

```
subroutine rhalf(obsDim,nrhs,y,Ry,t)
  use rdata
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: obsdim
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(obsDim,nrhs), intent(in) :: y
  real(kind=rk), dimension(obsDim,nrhs), intent(out) :: ry
  integer, intent(in) :: t
  ry = y*sqrt(0.1d0)
end subroutine RHALF
```

We also need to have the application of R^{-1} and $R^{-\frac{1}{2}}$. These can be done with the following subroutines:

```
subroutine solve_r(obsDim,nrhs,y,v,t)
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: obsdim
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(obsdim,nrhs), intent(in) :: y
  real(kind=rk), dimension(obsdim,nrhs), intent(out) :: v
  integer, intent(in) :: t
  v = y/0.1d0
end subroutine solve_r

subroutine solve_rhalf(obsdim,nrhs,y,v,t)
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: obsdim
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(obsdim,nrhs), intent(in) :: y
  real(kind=rk), dimension(obsdim,nrhs), intent(out) :: v
  integer, intent(in) :: t
  v = y/sqrt(0.1d0)
end subroutine solve_rhalf
```

7.1.3.2 Defining background error covariance matrix:

To make an initial ensemble, we can use a background error covariance matrix, B. In this example, $B = 0.2I$. There are two functions of this matrix that we could need: $B^{\frac{1}{2}}$ and B^{-1}. These can be coded in the following ways:

```
subroutine bhalf(nrhs,x,Qx)
  use sizes
  use qdata
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(state_dim,nrhs), intent(in) :: x
  real(kind=rk), dimension(state_dim,nrhs), intent(out) :: qx
  qx = sqrt(0.2d0)*x
end subroutine Bhalf

subroutine solve_b(nrhs,x,Qx)
  use sizes
  use qdata
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(state_dim,nrhs), intent(in) :: x
  real(kind=rk), dimension(state_dim,nrhs), intent(out) :: qx
  qx = x/0.2d0
end subroutine solve_b
```

7.1.3.3 Defining model error covariance matrix:

If i and j are two different grid points, then we define the correlation, C_{ij} between the model error at grid points x_i and x_j to be

$$C_{ij} = \begin{cases} 1 & \text{if } i = j \\ \frac{2}{3} & \text{if } |i-j| = 1 \\ \frac{1}{6} & \text{if } |i-j| = 2 \\ 0 & \text{otherwise} \end{cases}$$

Then, we define the model error covariance matrix $Q = \alpha^2 \frac{3}{2} C$. Hence

$$Q^{\frac{1}{2}} = \alpha \begin{bmatrix} 1 & 0.5 & 0 & 0 & \cdots & 0 & 0.5 \\ 0.5 & 1 & 0.5 & 0 & \cdots & 0 & 0 \\ 0 & 0.5 & 1 & 0.5 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0.5 \\ 0.5 & 0 & 0 & 0 & \cdots & 0.5 & 1 \end{bmatrix}$$

Thus this can be coded as:

```
subroutine qhalf(nrhs,x,Qx)
  use sizes
  use qdata
  implicit none
  integer, parameter :: rk=kind(1.0d+0)
  integer, intent(in) :: nrhs
  real(kind=rk), dimension(state_dim,nrhs), intent(in) :: x
  real(kind=rk), dimension(state_dim,nrhs), intent(out) :: qx
  real(kind=rk), parameter :: alpha=0.2
  integer :: i
  qx(1,:) = x(1,:) + 0.5d0*x(2,:) + 0.5d0*x(state_dim,:)
  qx(2:state_dim-1,:) = 0.5d0*x(1:state_dim-2,:)+x(2:state_dim-1,:)+0.5d0*x(3:state_dim,:)
  qx(state_dim,:) = 0.5*x(1,:) + 0.5*x(state_dim-1,:) + x(state_dim,:)
  qx = alpha*qx
end subroutine Qhalf
```

For simplicity, we shall leave the operator Q as the default one. This will simply apply Qhalf twice in order to apply the Q operator.

7.1.3.4 Specifying distance for localisation:

For the LETKF, we have to be able to do localisation. To do so, we define a distance measure between the observations and the grid points.

The model is cyclic, so we can say that all the variables lie in the interval [0,1]. To find the position of variable x_{xp} we can therefore divide xp by the total number of gridpoints. To find the position of observation y_{yp}, we note that y corresponds to every other gridpoint of x. Hence its position in the interval [0,1] can be calculated as $2yp-1$ divided by the total number of gridpoints. The distance between these two positions is then either the distance directly within the interval [0,1], or the distance wrapping around this interval. The code implementing this is below:

```
subroutine dist_st_ob(xp,yp,dis,t)
  use sizes
  implicit none
  integer, intent(in) :: xp
  integer, intent(in) :: yp
  real(kind=kind(1.0d0)), intent(out) :: dis
  integer, intent(in) :: t
  integer, parameter :: rk = kind(1.0d0)
  real(kind=rk) :: st,ob
  st = real(xp,rk)/real(state_dim,rk)
  ob = real((2*yp)-1,rk)/real(state_dim,rk)
  dis = min(abs(st-ob),1.0d0-abs(st-ob))
end subroutine dist_st_ob
```

7.1.3.5 Setting up configure model and reconfigure model for an experiment:

Here we tell EMPIRE how large the model is, how many observations we have per observation time. In this experiment we are going to have observations at a fixed frequency. The total number of observations in time, and the frequency of observations will be read in at run time to the variables `pf%time_obs` and `pf%time_bwn_obs`, respectively. Here we also call `timestep_data_set_total` to tell EMPIRE how long the model run will be. The "sanity check" below has helped me countless times when debugging - it serves no other purpose than to help identify errors.

```
subroutine configure_model
  use pf_control
  use timestep_data
```

```
  use sizes
  implicit none
  !this is for lorenz 96
  state_dim = 40
  obs_dim = 20
  call timestep_data_set_total(pf%time_bwn_obs*pf%time_obs)
  print*,'################################'
  print*,'######### SANITY CHECK ##########'
  print*,'################################'
  print*,'## STATE DIMENSION = ',state_dim
  print*,'##  OBS  DIMENSION = ',obs_dim
  print*,'################################'
end subroutine configure_model
```

In this example, the observational network is not going to change through time. Reconfigure model is called after each analysis is performed, so that the observational network can be reconfigured for the next set of observations. As the observation is going to be at the same time interval as for the last one, and the operators H and R remain constant, we this subroutine can be left blank as below. Note it cannot be removed as this will lead to a compilation error.

```
subroutine reconfigure_model
end subroutine reconfigure_model
```

7.1.3.6 Compiling the model specific changes

We are now at a point where we can compile the code. Go to the same directory as model_specific.f90 and simply type the command

```
1 make
```

Now you can check that a new `empire` executable was created. It will be found in the binaries folder bin/, run the following command and ensure that it has an up-to-date timestamp:

```
1 ls -l bin/empire
```

7.1.4 Running the codes

Now that we have both the model and EMPIRE compiled, we are in a position to execute the codes and therefore do some experiments. We shall do a twin experiment, where we first run the model to act as a "truth" and from which we generate observations. Then afterwards we will run an ensemble from different starting conditions and attempt to use the observations we have taken to stay close to the truth.

The first step is to run the truth and generate the observations.

7.1.4.1 Running the truth

We have to define the runtime parameters that EMPIRE requires. These are found in the file empire.nml.

- we want 1 observation in time: we set this in the variable time_obs
- we want the observations to occur every 4th model timestep: we set this in the variable time_bwn_obs
- we need to tell EMPIRE that it should be doing a truth run and generating the data: we set the logical variable gen_data to be true
- we need to tell EMPIRE how to initially perturb the model. We will do this as $x_t(0) = x_{\text{ref}}(0) + \eta$, where $\eta \sim \mathcal{N}(0, B)$: we set this in the variable init

In empire.nml, the namelist should appear as follows:

```
&pf_params
time_obs=1,
time_bwn_obs=4,
gen_data=.true.,
init='B'
/
```

Now let us move to an appropriate folder to run:

```
1 cd /path/to/where/you/want/to/run/the/code
```

In this folder, you *must* have the empire.nml file located. Check this with an `ls` command.

The model also needs a driving file. It needs to be called `l96.nml`, which is a standard Fortran namelist file. To set the parameters for the run we shall do, the l96.nml should look as follows:

```
&l96
N=40,
total_timesteps=4,
F=8.0d0,
dt=0.01
/
```

Now we want to run a single ensemble member. For this model, each ensemble member uses a single MPI process. So we must launch a total of 1 MPI processes to run the model. To output the truth, we only need a single EMPIRE process. The `mpirun` syntax for doing this is as follows:

```
1 mpirun -np 1 /path/to/model/executable : -np 1 /path/to/empire/executable
```

Now after the code has finished (a matter of seconds), let us look for some output. Check to see the observation files have been created:

```
1 ls obs*
```

There should only be one: `obs_000001`. This is going to be the observation file that we use in later.

7.1.4.2 Running a stochastic ensemble

Before we do the assimilation, let's get something to compare with. The comparison that we can do is against a model run where we have not done any assimilation.

We want EMPIRE to run for the same number of timesteps as before, so in empire.nml we set time_obs and time_bwn_obs as we had previously. We also want to create the initial ensemble in the same way, so init remains 'B'. We are no longer generating the data, so remove gen_data to use its default value false. Lastly, we have to tell EMPIRE to propagate the ensemble members forward stochastically without assimilating the observations. We do this by setting the filter type filter to 'SE' (Stochastic Ensemble).

In empire.nml, the namelist should appear as follows:

```
&pf_params
time_obs=1,
time_bwn_obs=4,
filter='SE',
init='B'
/
```

Now we want to execute this but using more than one ensemble member. 32 ensemble members seems like a good number. As we have more than 1 ensemble member, we can use more than one EMPIRE process. Here we will use 4, and therefore each EMPIRE process will be dealing with 8 ensemble members.

We run this with the `mpirun` command:

```
1 mpirun -np 32 /path/to/model/executable : -np 4 /path/to/empire/executable
```

7.1.4.3 Running an assimilation

All that is necessary to do in order to run an assimilation is now to change filter to correspond to the method we want to use. Let us use the LETKF (as we spent so long ensuring the distance calculation for it earlier).

Hence modify filter in empire.nml to 'LE' so that the namelist appears as:

```
&pf_params
time_obs=1,
time_bwn_obs=4,
filter='LE',
init='B'
/
```

Run this the same way as you ran the stochastic ensemble with the `mpirun` command:

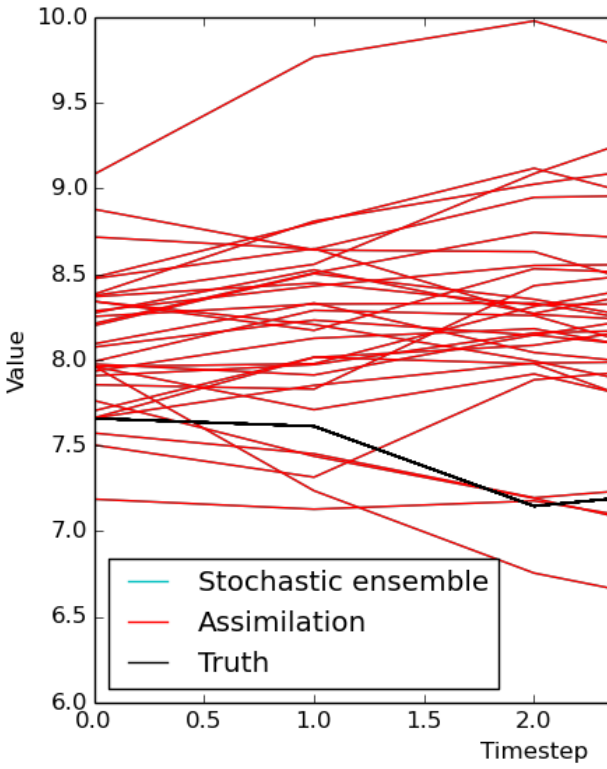
```
1 mpirun -np 32 /path/to/model/executable : -np 4 /path/to/empire/executable
```

7.1.5 Plotting the results

For this we shall use python, with numpy and matplotlib

The python script examples/lorenz96/tutorial_lorenz96_plot.py is able to produce some plots of trajectories from the output. Run this with the command

```
1 ../examples/lorenz96/tutorial_lorenz96_plot.py
```



If all has gone to plan, you should see a plot looking like the one below:
Notice how the ensemble members from the LETKF (in red, labelled "assimilation") narrows at timestep 4. This is where the observation occurred, and you should be able to see how the LETKF has brought the ensemble closer to the true state than the stochastic ensemble.

7.1.6 Tutorial codes

These can be found in the file `examples/lorenz96/tutorial_lorenz96_script.sh`

```
#!/bin/bash
set -o verbose

#make the lorenz96 code
make lorenz96

#backup the model_specific.f90 file
cp model_specific.f90 model_specific.f90_original

#make all the changes to model_specific.f90 that we have listed above
#happily they are in the file examples/lorenz96/model_specific_tutorial_l96.f90
cp examples/lorenz96/model_specific_tutorial_l96.f90 model_specific.f90

#build the codes
make

#check to see if the empire codes built properly
ls -l bin/empire

#make a run directory
mkdir -p rundirectory

#move to the run directory
cd rundirectory

#get the empire.nml file
cp ../examples/lorenz96/tutorial1.nml empire.nml

#look at the empire.nml file
cat empire.nml

#pause to look at this file
sleep 10

#generate the l96.nml file
echo -e "&l96\nN=40,\ntotal_timesteps=4,\nF=8.0d0,\ndt=1.0d-2\n/\n" > l96.nml

#look at the l96.nml file
cat l96.nml

#pause to look at this file
sleep 5

#generate the observations
mpirun --output-filename truth -np 1 ../bin/lorenz96 : -np 1 ../bin/empire

#look for the observation files
ls obs*

#modify the empire.nml file to run a stochastic ensemble
sed -i "s/filter.*/filter='SE',/g" empire.nml
sed -i "/gen_data/d" empire.nml

#look at the empire.nml file
cat empire.nml

#pause to look at this file
sleep 10

#now run the stochastic ensemble
mpirun --output-filename stoch -np 32 ../bin/lorenz96 : -np 4 ../bin/empire

#modify the empire.nml file to run the LETKF
sed -i "s/filter.*/filter='LE',/g" empire.nml

#look at the empire.nml file
cat empire.nml

#pause to look at this file
sleep 5

#now run the LETKF
mpirun --output-filename assim -np 32 ../bin/lorenz96 : -np 4 ../bin/empire

#plot the output
../examples/lorenz96/tutorial_lorenz96_plot.py
```

Chapter 8

Todo List

Page Assimilation Methods

Add some stuff about how to use this.

Type comms

Need to see what happens if some process has no observations in comms_v3

Fully document how to specify the model_as_subroutine calls in src/user/model

Subprogram diagnostics

test in anger with empire version 3. will probably segfault

Subprogram letkf_analysis

update to allow for non-diagonal R matrices to be used.

Subprogram letks_data::letks_filter_stage

update to allow for non-diagonal R matrices to be used.

Subprogram loc_function (loctype, dis, scal, inc)

include multiple localisation functions such as Gaspari-Cohn ones

Page Lorenz 96 Tutorial

Write some actual description of this model

Write some stuff about this. maybe a separate page.

Subprogram pf_control::pf_control_type::filter

change these to a longer string

Subprogram three_d_var (x)

make work with empire version 3

Subprogram threedvar_fcn (n, x, f, g)

update 3dvar to work with EMPIRE VERSION 3!

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

Data Type Documentation

11.1 comms Module Reference

Module containing EMPIRE coupling data.

Public Member Functions

- subroutine allocate_data
- subroutine deallocate_data
- subroutine initialise_mpi

  subroutine to select which mpi comms to use
- subroutine initialise_mpi_v1

  subroutine to make EMPIRE connections and saves details into pf_control module
- subroutine initialise_mpi_v2

  subroutine to initialise new version of empire
- subroutine initialise_mpi_v3

  subroutine to initialise even newer version of empire
- subroutine initialise_mpi_v4

  subroutine to initialise empire communicators when the model is to be a subroutine itself
- subroutine initialise_mpi_v5

  subroutine to initialise empire communication pattern similarly to v2 but with multiple ensemble members per model process
- subroutine send_all_models (stateDim, nrhs, x, tag)

  subroutine to send all the model states to the models
- subroutine recv_all_models (stateDim, nrhs, x)

  subroutine to receive all the model states from the models after
- subroutine irecv_all_models (stateDim, nrhs, x, requests)

  subroutine to receive all the model states from the models after
- subroutine verify_sizes

Public Attributes

- integer cpl_mpi_comm

  the communicator between the empire codes and the model master nodes
- integer world_rank

  the rank of this process on MPI_COMM_WORLD
- integer cpl_rank

the rank of this process on CPL_MPI_COMM

- integer nproc

  the total number of processes
- integer pf_mpi_comm

  the communicator between DA processes
- integer pfrank

  the rank of this process on PF_MPI_COMM
- integer npfs

  the total number of DA processes
- integer, dimension(:), allocatable gblcount

  the number of ensemble members associated with each DA process
- integer, dimension(:), allocatable gbldisp

  the displacements of each each ensemble member relative to pfrank=0. VERY useful for mpi_gatherv and mpi_scatterv on pf_mpi_comm
- integer nens

  the total number of ensemble members
- integer cnt

  the number of ensemble members associated with this process
- integer, dimension(:), allocatable particles

  the ensemble members associated with this process
- integer, dimension(:), allocatable cpl_mpi_comms

  communicators for if we are using empire v2 or v3
- integer, dimension(:), allocatable state_dims

  state dimensions on each model process for empire v2
- integer, dimension(:), allocatable state_displacements

  displacements of the various parts of the state vector for empire v2
- integer, dimension(:), allocatable obs_dims

  obs dimensions on each model process for empire v3
- integer, dimension(:), allocatable obs_displacements

  displacements of the various parts of the obs vector for empire v3
- integer mdl_num_proc

  number of processes of each ensemble member
- integer pf_member_comm

  communicator for empire v3 which contains all processes of individual ensemble members
- integer pf_ens_comm

  communicator for empire v3 which contains all ensemble members for that specific part of the state vector
- integer pf_ens_rank

  rank of the process on pf_ens_comm
- integer pf_ens_size

  size of pf_ens_comm for comms v3
- integer pf_member_rank

  rank of the process on pf_member_comm for empire v3
- integer pf_member_size

  size of pf_member_comm for empire v3
- integer, parameter comm_version =1

11.1.1 Detailed Description

Module containing EMPIRE coupling data.

Todo Need to see what happens if some process has no observations in comms_v3

11.1.2 comm_version

The integer parameter comm_version that is defined in comm_version.f90 defines the style of communication pattern used between the model and empire. There are currently 5 different patterns implemented:

- 1 = MPI SEND/RECV pairs between a single model process (single EMPIRE process per ensemble member)
- 2 = MPI GATHERV/SCATTERV between (possibly) multiple model processes (single EMPIRE process per ensemble member)
- 3 = MPI SEND/RECV pairs between multiple model processes and the same parallel process disribution in EMPIRE
- 4 = MODEL AS A SUBROUTINE OF EMPIRE

  Todo Fully document how to specify the model_as_subroutine calls in src/user/model
- 5 = Similar to 2, but with multiple ensemble members for each model process (TOMCAT CASE)

For more information, see the pages Communication Methods and EMPIRE communicators for more information.

Definition at line 57 of file comms.f90.

11.1.3 Member Function/Subroutine Documentation

11.1.3.1 subroutine comms::allocate_data ( )

Definition at line 106 of file comms.f90.

11.1.3.2 subroutine comms::deallocate_data ( )

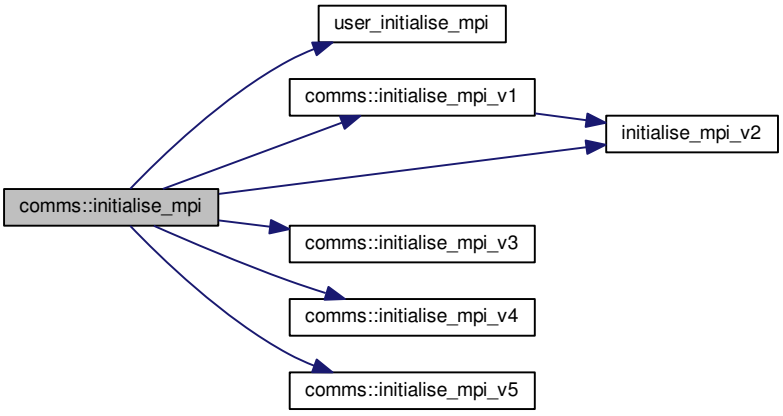
Definition at line 112 of file comms.f90.

11.1.3.3 subroutine comms::initialise_mpi ( )

subroutine to select which mpi comms to use

Definition at line 118 of file comms.f90.

Here is the call graph for this function:

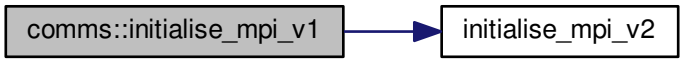


11.1.3.4 subroutine comms::initialise_mpi_v1 ( )

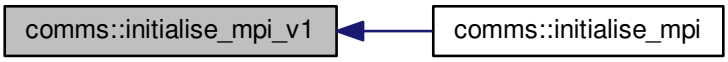
subroutine to make EMPIRE connections and saves details into pf_control module

Definition at line 145 of file comms.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



11.1.3.5 subroutine comms::initialise_mpi_v2 ( )

subroutine to initialise new version of empire

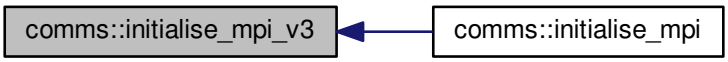
Definition at line 226 of file comms.f90.

11.1.3.6 subroutine comms::initialise_mpi_v3 ( )

subroutine to initialise even newer version of empire

Definition at line 397 of file comms.f90.

Here is the caller graph for this function:

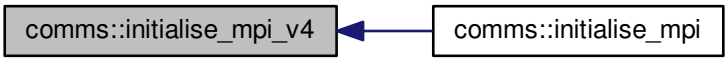


11.1.3.7 subroutine comms::initialise_mpi_v4 ( )

subroutine to initialise empire communicators when the model is to be a subroutine itself

Definition at line 606 of file comms.f90.

Here is the caller graph for this function:

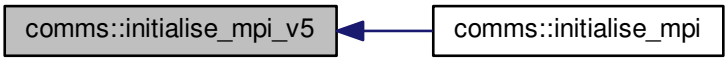


11.1.3.8 subroutine comms::initialise_mpi_v5 ( )

subroutine to initialise empire communication pattern similarly to v2 but with multiple ensemble members per model process

Definition at line 717 of file comms.f90.

Here is the caller graph for this function:

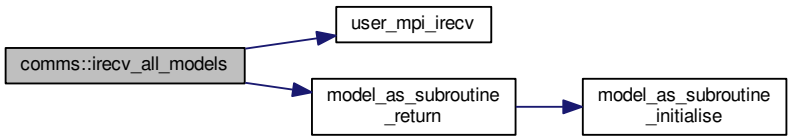


11.1.3.9 subroutine comms::irecv_all_models (integer, intent(in) *stateDim,* integer, intent(in) *nrhs,* real(kind=kind(1.0d0)), dimension(statedim,nrhs), intent(out) *x,* integer, dimension(nrhs), intent(inout) *requests*)

subroutine to receive all the model states from the models after

Definition at line 1038 of file comms.f90.

Here is the call graph for this function:

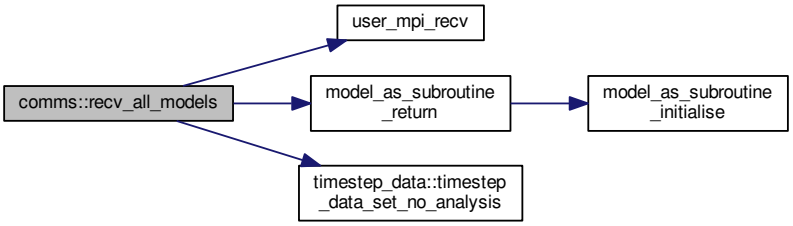


11.1.3.10 subroutine comms::recv_all_models (integer, intent(in) *stateDim,* integer, intent(in) *nrhs,* real(kind=kind(1.0d0)), dimension(statedim,nrhs), intent(out) *x*)

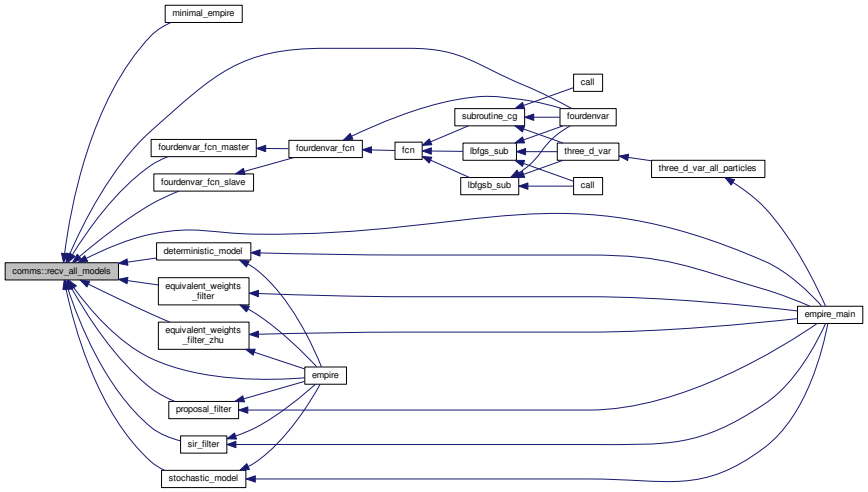
subroutine to receive all the model states from the models after

Definition at line 986 of file comms.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

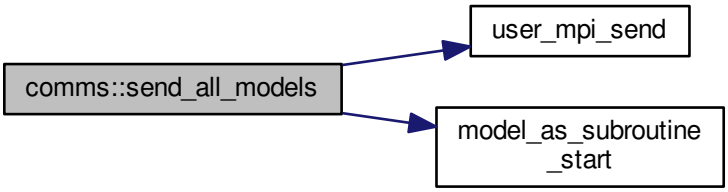


11.1.3.11 subroutine comms::send_all_models (integer, intent(in) *stateDim,* integer, intent(in) *nrhs,* real(kind=kind(1.0d0)), dimension(statedim,nrhs), intent(in) *x,* integer, intent(in) *tag*)

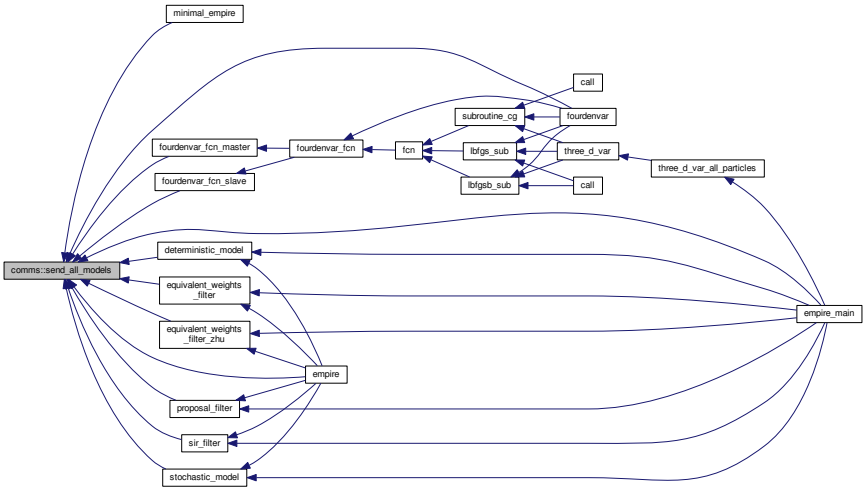
subroutine to send all the model states to the models

Definition at line 938 of file comms.f90.

Here is the call graph for this function:



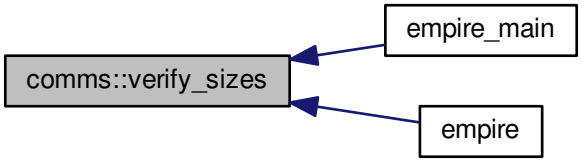
Here is the caller graph for this function:



11.1.3.12 subroutine comms::verify_sizes ( )

Definition at line 1101 of file comms.f90.

Here is the caller graph for this function:



11.1.4 Member Data Documentation

11.1.4.1 integer comms::cnt

the number of ensemble members associated with this process

Definition at line 74 of file comms.f90.

11.1.4.2 integer, parameter comm_version =1

The style of communication between the model and empire. See comm_version for an up-to-date description of the options implemented

Definition at line 41 of file comm_version.f90.

11.1.4.3 integer comms::cpl_mpi_comm

the communicator between the empire codes and the model master nodes

Definition at line 60 of file comms.f90.

11.1.4.4 integer, dimension(:), allocatable comms::cpl_mpi_comms

communicators for if we are using empire v2 or v3

Definition at line 78 of file comms.f90.

11.1.4.5 integer comms::cpl_rank

the rank of this process on CPL_MPI_COMM

Definition at line 63 of file comms.f90.

11.1.4.6 integer, dimension(:), allocatable comms::gblcount

the number of ensemble members associated with each DA process

Definition at line 68 of file comms.f90.

11.1.4.7 integer, dimension(:), allocatable comms::gbldisp

the displacements of each each ensemble member relative to pfrank=0. VERY useful for mpi_gatherv and mpi_↩ scatterv on pf_mpi_comm

Definition at line 70 of file comms.f90.

11.1.4.8 integer comms::mdl_num_proc

number of processes of each ensemble member

Definition at line 90 of file comms.f90.

11.1.4.9 integer comms::nens

the total number of ensemble members

Definition at line 73 of file comms.f90.

11.1.4.10 integer comms::npfs

the total number of DA processes
Definition at line 67 of file comms.f90.

11.1.4.11 integer comms::nproc

the total number of processes
Definition at line 64 of file comms.f90.

11.1.4.12 integer, dimension(:), allocatable comms::obs_dims

obs dimensions on each model process for empire v3
Definition at line 85 of file comms.f90.

11.1.4.13 integer, dimension(:), allocatable comms::obs_displacements

displacements of the various parts of the obs vector for empire v3
Definition at line 87 of file comms.f90.

11.1.4.14 integer, dimension(:), allocatable comms::particles

the ensemble members associated with this process
Definition at line 76 of file comms.f90.

11.1.4.15 integer comms::pf_ens_comm

communicator for empire v3 which contains all ensemble members for that specific part of the state vector
Definition at line 95 of file comms.f90.

11.1.4.16 integer comms::pf_ens_rank

rank of the process on pf_ens_comm
Definition at line 98 of file comms.f90.

11.1.4.17 integer comms::pf_ens_size

size of pf_ens_comm for comms v3
Definition at line 99 of file comms.f90.

11.1.4.18 integer comms::pf_member_comm

communicator for empire v3 which contains all processes of individual ensemble members
Definition at line 92 of file comms.f90.

11.1.4.19 integer comms::pf_member_rank

rank of the process on pf_member_comm for empire v3

Definition at line 100 of file comms.f90.

11.1.4.20 integer comms::pf_member_size

size of pf_member_comm for empire v3

Definition at line 102 of file comms.f90.

11.1.4.21 integer comms::pf_mpi_comm

the communicator between DA processes

Definition at line 65 of file comms.f90.

11.1.4.22 integer comms::pfrank

the rank of this process on PF_MPI_COMM

Definition at line 66 of file comms.f90.

11.1.4.23 integer, dimension(:), allocatable comms::state_dims

state dimensions on each model process for empire v2

Definition at line 80 of file comms.f90.

11.1.4.24 integer, dimension(:), allocatable comms::state_displacements

displacements of the various parts of the state vector for empire v2

Definition at line 82 of file comms.f90.

11.1.4.25 integer comms::world_rank

the rank of this process on MPI_COMM_WORLD

Definition at line 62 of file comms.f90.

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

- src/utils/comms.f90
- comm_version.f90

11.2 communicator_version Module Reference

module to store the parameter comm_version to control the communication pattern that empire will use.

11.2.1 Detailed Description

module to store the parameter comm_version to control the communication pattern that empire will use.

this should be set by the user before compilation so that the correct communicator version is used. see comm_version for an up-to-date description of the options for this.

This file is not tracked by git, so any changes that the user makes here will not be updated by a *git pull* command

Definition at line 40 of file comm_version.f90.

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

- comm_version.f90

11.3 compile_options Module Reference

Module that stores logical variables to control the compilation.

Public Attributes

- logical opt_petsc

  Compile option to use PETSC.

11.3.1 Detailed Description

Module that stores logical variables to control the compilation.

Definition at line 29 of file compile_options.f90.

11.3.2 Member Data Documentation

11.3.2.1 logical compile_options::opt_petsc

Compile option to use PETSC.

Definition at line 31 of file compile_options.f90.

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

- src/controllers/compile_options.f90

11.4 fourdenvardata Module Reference

module holding data specific for 4denvar, not var itself. this is necessary because of the difference in x in optimization and in the model state.

Public Member Functions

- subroutine allocate4denvardata
- subroutine read_background_term ()

  subroutine to read xb from file
- subroutine deallocate4denvardata
- subroutine read_ensemble_perturbation_matrix

  subroutine to read in the ensemble perturbation matrix

Public Attributes

- integer m
  the number of perturbations, or nens-1
- real(kind=kind(1.0d0)), dimension(:), allocatable xb
  the background guess
- real(kind=kind(1.0d0)), dimension(:,:), allocatable x0
  the initial ensemble perturbation matrix
- real(kind=kind(1.0d0)), dimension(:,:), allocatable xt
  the current ensemble

11.4.1 Detailed Description

module holding data specific for 4denvar, not var itself. this is necessary because of the difference in x in optimization and in the model state.

Definition at line 32 of file fourdenvardata.f90.

11.4.2 Member Function/Subroutine Documentation

11.4.2.1 subroutine fourdenvardata::allocate4denvardata ( )

Definition at line 44 of file fourdenvardata.f90.

Here is the caller graph for this function:

11.4.2.2 subroutine fourdenvardata::deallocate4denvardata ( )

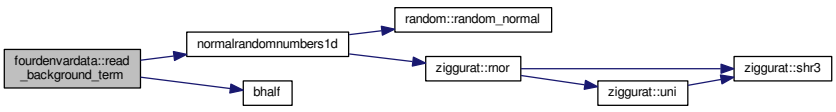
Definition at line 80 of file fourdenvardata.f90.

11.4.2.3 subroutine fourdenvardata::read_background_term ( )

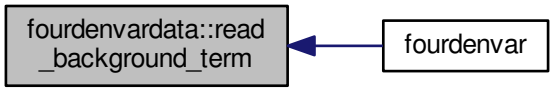
subroutine to read xb from file

Definition at line 62 of file fourdenvardata.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



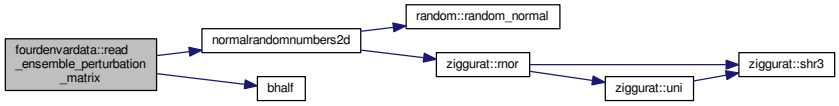
11.4.2.4 subroutine fourdenvardata::read_ensemble_perturbation_matrix ( )

subroutine to read in the ensemble perturbation matrix

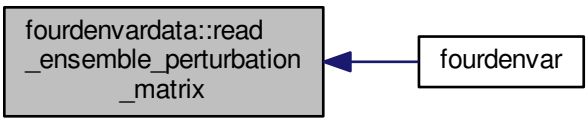
we need to fill in the entries of x0 here

Definition at line 89 of file fourdenvardata.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



11.4.3 Member Data Documentation

11.4.3.1 integer fourdenvardata::m

the number of perturbations, or nens-1

Definition at line 34 of file fourdenvardata.f90.

11.4.3.2 real(kind=kind(1.0d0)), dimension(:,:), allocatable fourdenvardata::x0

the initial ensemble perturbation matrix

THIS IS ***NOT*** SCALED!!

Definition at line 37 of file fourdenvardata.f90.

11.4.3.3 real(kind=kind(1.0d0)), dimension(:), allocatable fourdenvardata::xb

the background guess

Definition at line 35 of file fourdenvardata.f90.

11.4.3.4 real(kind=kind(1.0d0)), dimension(:,:), allocatable fourdenvardata::xt

the current ensemble

Definition at line 41 of file fourdenvardata.f90.

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

- src/4dEnVar/fourdenvardata.f90

11.5 histogram_data Module Reference

Module to control what variables are used to generate rank histograms.

Public Member Functions

- subroutine load_histogram_data

  subroutine to read from variables_hist.dat which holds the variables to be used to make the rank histograms
- subroutine kill_histogram_data

  subroutine to clean up arrays used in rank histograms

Public Attributes

- integer, dimension(:), allocatable rank_hist_list
- integer, dimension(:), allocatable rank_hist_nums
- integer rhl_n
- integer rhn_n

11.5.1 Detailed Description

Module to control what variables are used to generate rank histograms.

Definition at line 29 of file histogram.f90.

11.5.2 Member Function/Subroutine Documentation

11.5.2.1 subroutine histogram_data::kill_histogram_data ( )

subroutine to clean up arrays used in rank histograms

Definition at line 135 of file histogram.f90.

11.5.2.2 subroutine histogram_data::load_histogram_data ( )

subroutine to read from variables_hist.dat which holds the variables to be used to make the rank histograms

In order for histograms to be output, the file "variables_hist.dat" must contain the following infomation:

- rhn_n – the number of different rank histograms to be output
- the numbers of variables to be included in each rank histogram
- the index of the state vector for each different variable in each different rank histogram, grouped by the different histograms

So as an example, suppose we wanted to produce 3 rank histograms, the first relating to the 10th, and 16th variables in the state vector, the second containing the 1st, 2nd, 56th and 98th variables of the state vector and the final rank histogram relating to the 6th, 11th, 19th, 45th and 32nd variables. Then variables_hist.dat would look as follows:

```
3
2
4
5
10
16
1
2
56
98
6
11
19
45
32
```

Definition at line 73 of file histogram.f90.

11.5.3 Member Data Documentation

11.5.3.1 integer, dimension(:), allocatable histogram_data::rank_hist_list

Definition at line 30 of file histogram.f90.

11.5.3.2 integer, dimension(:), allocatable histogram_data::rank_hist_nums

Definition at line 31 of file histogram.f90.

11.5.3.3 integer histogram_data::rhl_n

Definition at line 32 of file histogram.f90.

11.5.3.4 integer histogram_data::rhn_n

Definition at line 32 of file histogram.f90.

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

- src/utils/histogram.f90

11.6 hqht_plus_r Module Reference

Public Member Functions

- subroutine load_hqhtr
- subroutine hqhtr_factor
- subroutine kill_hqhtr

11.6.1 Detailed Description

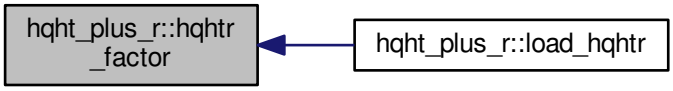
Definition at line 44 of file Rdata.f90.

11.6.2 Member Function/Subroutine Documentation

11.6.2.1 subroutine hqht_plus_r::hqhtr_factor ( )

Definition at line 54 of file Rdata.f90.

Here is the caller graph for this function:



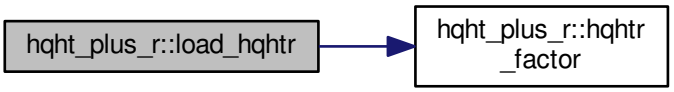
11.6.2.2 subroutine hqht_plus_r::kill_hqhtr ( )

Definition at line 59 of file Rdata.f90.

11.6.2.3 subroutine hqht_plus_r::load_hqhtr ( )

Definition at line 50 of file Rdata.f90.

Here is the call graph for this function:



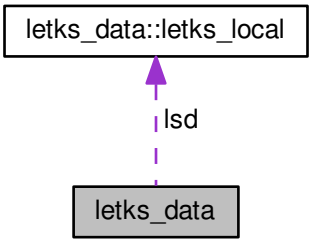
The documentation for this module was generated from the following file:

- src/user/Rdata.f90

11.7 letks_data Module Reference

module for doing things related to the LETKS:

Collaboration diagram for letks_data:



Data Types

- type letks_local

Public Member Functions

- subroutine allocate_letks (N)
- subroutine deallocate_letks ()
- subroutine letks_filter_stage

  subroutine to compute the data for the LETKS, so that the increments can subsquently be computed
- subroutine letks_increment (psi, inc)

  subroutine to compute the LETKS increments

Public Attributes

- type(letks_local), dimension(:), allocatable lsd

11.7.1 Detailed Description

module for doing things related to the LETKS:

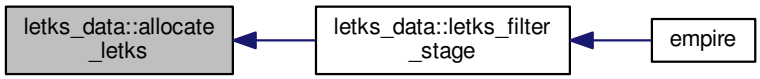
Definition at line 31 of file letks.f90.

11.7.2 Member Function/Subroutine Documentation

11.7.2.1 subroutine letks_data::allocate_letks (integer, intent(in) *N*)

Definition at line 42 of file letks.f90.

Here is the caller graph for this function:



11.7.2.2 subroutine letks_data::deallocate_letks ( )

Definition at line 47 of file letks.f90.

11.7.2.3 subroutine letks_data::letks_filter_stage ( )

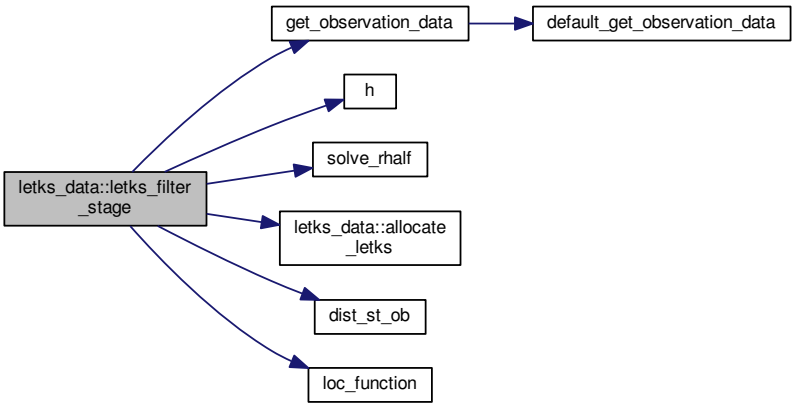
subroutine to compute the data for the LETKS, so that the increments can subsquently be computed

Todo update to allow for non-diagonal R matrices to be used.

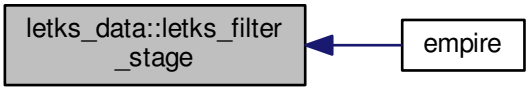
The observation

Definition at line 54 of file letks.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.2.4 subroutine letks_data::letks_increment (real(kind=rk), dimension(state_dim,pf%count), intent(in) *psi*, real(kind=rk), dimension(state_dim,pf%count), intent(out) *inc*)

subroutine to compute the LETKS increments

Parameters

in	*psi*	input ensemble
out	*inc*	LETKS increment

Definition at line 394 of file letks.f90.

Here is the caller graph for this function:

11.7.3 Member Data Documentation

11.7.3.1 type (letks_local), dimension(:), allocatable letks_data::lsd

Definition at line 39 of file letks.f90.

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

- src/smoothers/letks.f90

11.8 letks_data::letks_local Type Reference

Public Attributes

- integer red_obsdim
- real(kind=kind(1.0d0)), dimension(:,:), allocatable usiut
- real(kind=kind(1.0d0)), dimension(:), allocatable ud

11.8.1 Detailed Description

Definition at line 33 of file letks.f90.

11.8.2 Member Data Documentation

11.8.2.1 integer letks_data::letks_local::red_obsdim

Definition at line 34 of file letks.f90.

11.8.2.2 real(kind=kind(1.0d0)), dimension(:), allocatable letks_data::letks_local::ud

Definition at line 36 of file letks.f90.

11.8.2.3 real(kind=kind(1.0d0)), dimension(:,:), allocatable letks_data::letks_local::usiut

Definition at line 35 of file letks.f90.

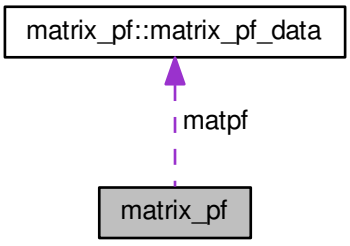
The documentation for this type was generated from the following file:

- src/smoothers/letks.f90

11.9 matrix_pf Module Reference

module to deal with generating and outputting pf matrix

Collaboration diagram for matrix_pf:



Data Types

- type matrix_pf_data

Public Member Functions

- subroutine read_matrix_pf_information
  subroutine to read namelist to control this output
- subroutine matrix_pf_output (root, comm, n, m, x, time, is_analysis)
  subroutine to generate and output matrix Pf

Public Attributes

- type(matrix_pf_data), save matpf

  module holding data for generating and outputting P_f matrix.
  Note: this feature is not accessible with empire version 3 communications; the matrices in question are simply too large to compute the full P_f matrix.

11.9.1 Detailed Description

module to deal with generating and outputting pf matrix

Definition at line 29 of file matrix_pf.f90.

11.9.2 Member Function/Subroutine Documentation

11.9.2.1 subroutine matrix_pf::matrix_pf_output (integer, intent(in) *root,* integer, intent(in) *comm,* integer, intent(in) *n,* integer, intent(in) *m,* real(kind=rk), dimension(n,m) *x,* integer, intent(in) *time,* logical, intent(in) *is_analysis*)

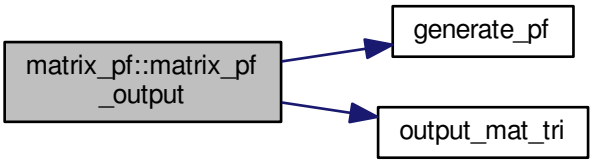
subroutine to generate and output matrix Pf

Parameters

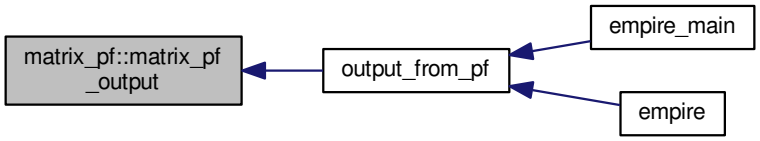
in	*root*	the process to output file
in	*comm*	the mpi communicator to build the matrix on
in	*n*	the size of the state vector
in	*m*	the number of state vectors on this process
	x	the local ensemble members
in	*time*	the current timestep
in	*is_analysis*	true if analysis just performed

Definition at line 116 of file matrix_pf.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

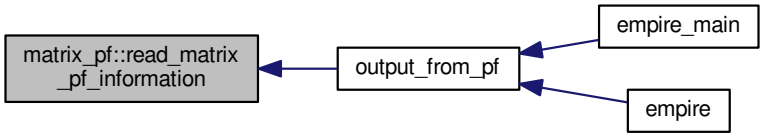


11.9.2.2 subroutine matrix_pf::read_matrix_pf_information ( )

subroutine to read namelist to control this output

Definition at line 59 of file matrix_pf.f90.

Here is the caller graph for this function:



11.9.3 Member Data Documentation

11.9.3.1 type(matrix_pf_data), save matrix_pf::matpf

module holding data for generating and outputting P_f matrix.
Note: this feature is not accessible with empire version 3 communications; the matrices in question are simply too large to compute the full P_f matrix.

Definition at line 52 of file matrix_pf.f90.

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

- src/utils/matrix_pf.f90

11.10 matrix_pf::matrix_pf_data Type Reference

Public Attributes

- character(30) prefix

  the prefix of the filename to be output
- integer k

  the frequency to output the matrix

- logical analysis
  if true, output at all analysis times
- logical frequency
  if true, output at all timesteps that are 0 mod k
- integer output_type
  output file type.

11.10.1 Detailed Description

Definition at line 31 of file matrix_pf.f90.

11.10.2 Member Data Documentation

11.10.2.1 logical matrix_pf::matrix_pf_data::analysis

if true, output at all analysis times

Definition at line 38 of file matrix_pf.f90.

11.10.2.2 logical matrix_pf::matrix_pf_data::frequency

if true, output at all timesteps that are 0 mod k

Definition at line 39 of file matrix_pf.f90.

11.10.2.3 integer matrix_pf::matrix_pf_data::k

the frequency to output the matrix

Definition at line 37 of file matrix_pf.f90.

11.10.2.4 integer matrix_pf::matrix_pf_data::output_type

output file type.

- 0 - undefined
- 1 - standard packed format (TP)
- 2 - rectangular full packed format (TF)
  Negative values will be formatted.
  Positive values will be unformatted.

Definition at line 41 of file matrix_pf.f90.

11.10.2.5 character(30) matrix_pf::matrix_pf_data::prefix

the prefix of the filename to be output

Definition at line 35 of file matrix_pf.f90.

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

- src/utils/matrix_pf.f90

11.11 model_as_subroutine_data Module Reference

a module that can be used to store the data for when the model is a subroutine of empire, i.e. using comms_version 4

Public Attributes

- logical, save initialised =.false.
- integer num_of_ensemble_members
- integer first_ptcl
- integer final_ptcl
- real(kind=kind(1.0d0)), dimension(:,:), allocatable model_states

11.11.1 Detailed Description

a module that can be used to store the data for when the model is a subroutine of empire, i.e. using comms_version 4

Definition at line 30 of file model_as_subroutine_data.f90.

11.11.2 Member Data Documentation

11.11.2.1 integer model_as_subroutine_data::final_ptcl

Definition at line 35 of file model_as_subroutine_data.f90.

11.11.2.2 integer model_as_subroutine_data::first_ptcl

Definition at line 34 of file model_as_subroutine_data.f90.

11.11.2.3 logical, save model_as_subroutine_data::initialised =.false.

Definition at line 32 of file model_as_subroutine_data.f90.

11.11.2.4 real(kind=kind(1.0d0)), dimension(:,:), allocatable model_as_subroutine_data::model_states

Definition at line 36 of file model_as_subroutine_data.f90.

11.11.2.5 integer model_as_subroutine_data::num_of_ensemble_members

Definition at line 33 of file model_as_subroutine_data.f90.

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

- src/user/model/model_as_subroutine_data.f90

11.12 output_empire Module Reference

Module that stores the information about the outputting from empire.

Public Member Functions

- subroutine open_emp_o (id_num)

  subroutine to open the file for outputting
- subroutine close_emp_o ()

  subroutine to close the output file

Public Attributes

- integer, parameter emp_o =6

  the output stream number
- integer, parameter unit_nml =10

  the unit number for reading empire.nml
- integer, parameter unit_obs =11

  the unit number for reading and writing observations
- integer, parameter unit_truth =12

  the unit number for reading and writing the truth
- integer, parameter unit_weight =13

  the unit number for writing the ensemble weights
- integer, parameter unit_mean =14

  the unit number for writing the ensemble mean
- integer, parameter unit_state =15

  the unit number for reading and writing the state
- integer, parameter unit_ens_rmse =16

  the unit number for reading and writing the ensemble rmse
- integer, parameter unit_mat_tri =17

  the unit number for outputing triangular matrices
- integer, parameter unit_spatial_rmse =18

  the unit number for writing the spatial rmse
- integer, parameter unit_variance =19

  the unit number for outputing the ensemble variance
- integer, parameter unit_hist_read =20

  the unit number for reading histogram data
- integer, parameter unit_hist_write =21

  the unit number for writing histogram data
- integer, parameter unit_hist_readt =22

  the unit number for reading histogram truth data
- integer, parameter unit_hist_readp =23

  the unit number for reading histogram particle data
- integer, parameter unit_traj_read =24

  the unit number for reading trajectory data
- integer, parameter unit_traj_write =24

  the unit number for writing trajectory data
- integer, parameter unit_vardata =25

  the unit number for reading vardata

11.12.1 Detailed Description

Module that stores the information about the outputting from empire.

Definition at line 30 of file output_empire.f90.

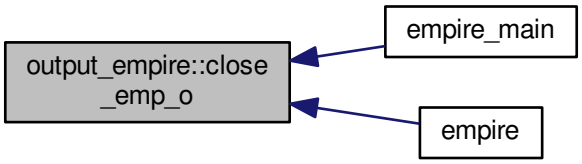
11.12.2 Member Function/Subroutine Documentation

11.12.2.1 subroutine output_empire::close_emp_o ( )

subroutine to close the output file

Definition at line 137 of file output_empire.f90.

Here is the caller graph for this function:



11.12.2.2 subroutine output_empire::open_emp_o (integer, intent(in) *id_num*)

subroutine to open the file for outputting

in order to redirect the STDOUT used by EMPIRE, this subroutine will read from the file 'empire.nml'. If it exists, it looks for the namelist &empire_output, which consists of a single string up to 10 characters called 'basename' which will be read, and the STDOUT redirected to that string appended with the MPI rank of the EMPIRE process.

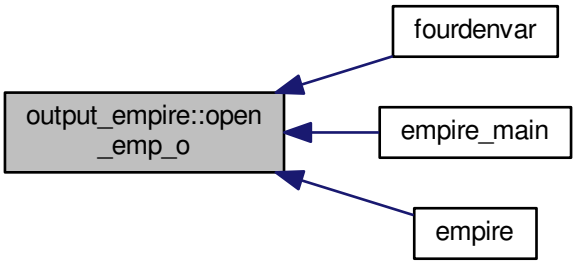
In order to suppress most of the STDOUT from EMPIRE, this path can be set to a platform specific Null device:

- Unix: /dev/null
- MS: nul

If you are running on any other system, please let me know what Null Device you would like to use, and we can add a check for it

Definition at line 84 of file output_empire.f90.

Here is the caller graph for this function:



11.12.3 Member Data Documentation

11.12.3.1 integer, parameter output_empire::emp_o =6

the output stream number

Definition at line 32 of file output_empire.f90.

11.12.3.2 integer, parameter output_empire::unit_ens_rmse =16

the unit number for reading and writing the ensemble rmse

Definition at line 44 of file output_empire.f90.

11.12.3.3 integer, parameter output_empire::unit_hist_read =20

the unit number for reading histogram data

Definition at line 52 of file output_empire.f90.

11.12.3.4 integer, parameter output_empire::unit_hist_readp =23

the unit number for reading histogram particle data

Definition at line 58 of file output_empire.f90.

11.12.3.5 integer, parameter output_empire::unit_hist_readt =22

the unit number for reading histogram truth data

Definition at line 56 of file output_empire.f90.

11.12.3.6 integer, parameter output_empire::unit_hist_write =21

the unit number for writing histogram data

Definition at line 54 of file output_empire.f90.

11.12.3.7 integer, parameter output_empire::unit_mat_tri =17

the unit number for outputing triangular matrices

Definition at line 46 of file output_empire.f90.

11.12.3.8 integer, parameter output_empire::unit_mean =14

the unit number for writing the ensemble mean

Definition at line 40 of file output_empire.f90.

11.12.3.9 integer, parameter output_empire::unit_nml =10

the unit number for reading empire.nml

Definition at line 33 of file output_empire.f90.

11.12.3.10 integer, parameter output_empire::unit_obs =11

the unit number for reading and writing observations

Definition at line 34 of file output_empire.f90.

11.12.3.11 integer, parameter output_empire::unit_spatial_rmse =18

the unit number for writing the spatial rmse

Definition at line 48 of file output_empire.f90.

11.12.3.12 integer, parameter output_empire::unit_state =15

the unit number for reading and writing the state

Definition at line 42 of file output_empire.f90.

11.12.3.13 integer, parameter output_empire::unit_traj_read =24

the unit number for reading trajectory data

Definition at line 60 of file output_empire.f90.

11.12.3.14 integer, parameter output_empire::unit_traj_write =24

the unit number for writing trajectory data

Definition at line 62 of file output_empire.f90.

11.12.3.15 integer, parameter output_empire::unit_truth =12

the unit number for reading and writing the truth

Definition at line 36 of file output_empire.f90.

11.12.3.16 integer, parameter output_empire::unit_vardata =25

the unit number for reading vardata

Definition at line 64 of file output_empire.f90.

11.12.3.17 integer, parameter output_empire::unit_variance =19

the unit number for outputing the ensemble variance

Definition at line 50 of file output_empire.f90.

11.12.3.18 integer, parameter output_empire::unit_weight =13

the unit number for writing the ensemble weights

Definition at line 38 of file output_empire.f90.

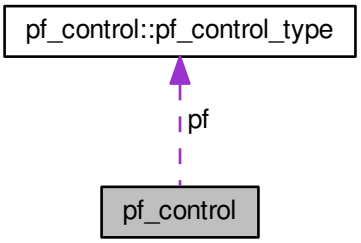
The documentation for this module was generated from the following file:

- src/controllers/output_empire.f90

11.13 pf_control Module Reference

module pf_control holds all the information to control the the main program

Collaboration diagram for pf_control:



Data Types

- type pf_control_type

Public Member Functions

- subroutine set_pf_controls

  subroutine to ensure pf_control data is ok
- subroutine parse_pf_parameters

  subroutine to read the namelist file and save it to pf datatype Here we read pf_parameters.dat or empire.nml
- subroutine deallocate_pf

  subroutine to deallocate space for the filtering code

Public Attributes

- type(pf_control_type), save pf

  the derived data type holding all controlling data

11.13.1 Detailed Description

module pf_control holds all the information to control the the main program

Definition at line 29 of file pf_control.f90.

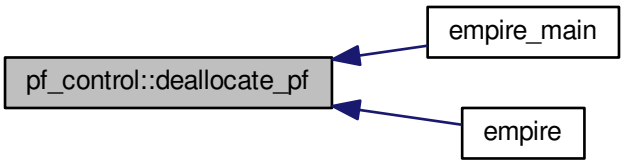
11.13.2 Member Function/Subroutine Documentation

11.13.2.1 subroutine pf_control::deallocate_pf ( )

subroutine to deallocate space for the filtering code

Definition at line 451 of file pf_control.f90.

Here is the caller graph for this function:



11.13.2.2 subroutine pf_control::parse_pf_parameters ( )

subroutine to read the namelist file and save it to pf datatype Here we read pf_parameters.dat or empire.nml pf_parameters.dat or empire.nml is a fortran namelist file. As such, within it there must be a line beginning &pf_params

To make it (probably) work, ensure there is a forward slash on the penultimate line and a blank line to end the file This is just the fortran standard for namelists though.

On to the content...in any order, the pf_parameters.dat (or empire.nml) file may contain the following things:

Integers:

- time_obs
- time_bwn_obs

Reals, double precision:

- nudgefac
- nfac
- ufac
- Qscale
- keep
- rho
- len

2 Characters:

- filter

1 Character:

- init

Logicals:

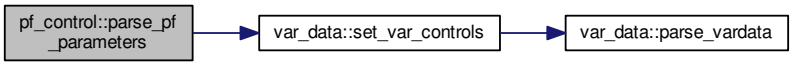
- gen_Q
- gen_data
- use_talagrand
- use_mean
- use_variance
- use_traj
- use_spatial_rmse
- use_ens_rmse
- output_weights

250 Character string:

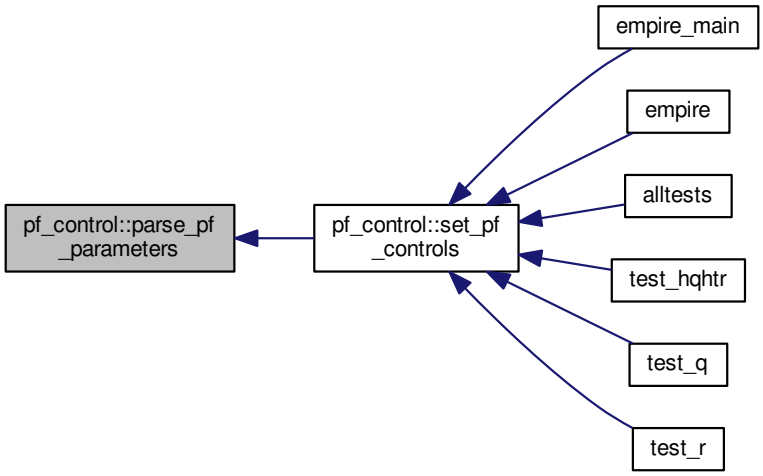
- rmse_filename

Definition at line 193 of file pf_control.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



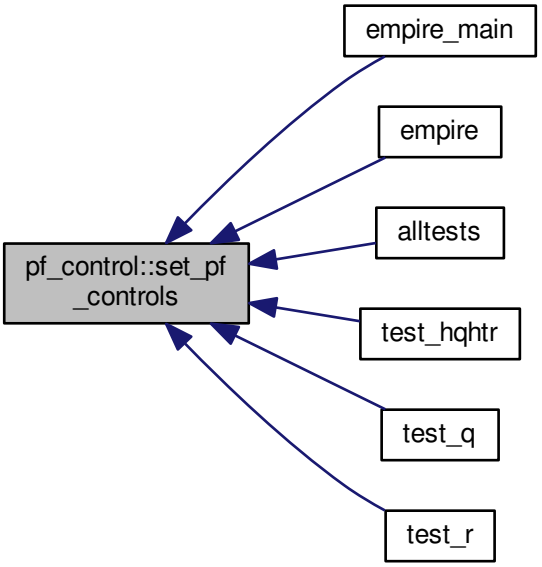
11.13.2.3 subroutine pf_control::set_pf_controls ( )

subroutine to ensure pf_control data is ok

Definition at line 125 of file pf_control.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



11.13.3 Member Data Documentation

11.13.3.1 type(pf_control_type), save pf_control::pf

the derived data type holding all controlling data

Definition at line 120 of file pf_control.f90.

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

- src/controllers/pf_control.f90

11.14 pf_control::pf_control_type Type Reference

Public Attributes

- integer nens

  the total number of ensemble members
- real(kind=kind(1.0d0)), dimension(:), allocatable weight

  the negative log of the weights of the particles
- integer time_obs

  the number of observations we will assimilate
- integer time_bwn_obs

  the number of model timesteps between observations
- real(kind=kind(1.0d0)) nudgefac

  the nudging factor
- logical gen_data

  true generates synthetic obs for a twin experiment
- logical gen_q

  true attempts to build up Q from long model run. UNUSED. DOES NOTHING!
- integer timestep =0

  the current timestep as the model progresses
- real(kind=kind(1.0d0)), dimension(:,:), allocatable psi

  state vector of ensemble members on this mpi process
- real(kind=kind(1.0d0)), dimension(:), allocatable mean

  mean state vector
- real(kind=kind(1.0d0)) nfac

  standard deviation of normal distribution in mixture density
- real(kind=kind(1.0d0)) ufac

  half width of the uniform distribution in mixture density
- real(kind=kind(1.0d0)) efac
- real(kind=kind(1.0d0)) keep

  proportion of particles to keep in EWPF EW step
- real(kind=kind(1.0d0)) time

  dunno
- real(kind=kind(1.0d0)) qscale

  scalar to multiply Q by
- real(kind=kind(1.0d0)) rho

  enkf inflation factor so that $P_f = (1+\rho)^2 P_f$
- real(kind=kind(1.0d0)) len

  R localisation length scale The entries in the observation error covariance matrix R are multiplied by the function $\exp\left(\frac{\text{dist}^2}{2\text{len}^2}\right)$.
- integer couple_root

  empire master processor
- logical use_talagrand

  switch if true outputs rank histograms. See load_histogram_data for details.
- logical output_weights

  switch if true outputs ensemble weights
- logical use_mean

  switch if true outputs ensemble mean
- logical use_variance

  switch if true outputs ensemble variance
- logical use_traj

  switch if true outputs trajectories
- logical use_spatial_rmse

switch if true outputs Root Mean Square Errors
See Outputting Root Mean Squared Errors for more information

- logical use_ens_rmse

  switch if true outputs the field of root mean squared errors where $rmse(j) = \sqrt{\frac{1}{N_e}\sum_{i=1}^{N_e}(x_i(j) - x^t(j))^2}$

- character(250) rmse_filename

  string to hold the name of the file to output rmse to

- integer, dimension(:,:), allocatable talagrand

  storage for rank histograms

- integer count

  number of ensemble members associated with this MPI process

- integer, dimension(:), allocatable particles

  particles associates with this MPI process

- character(2) filter

  which filter to use currently this has a number of options:

- character(1) init

  which method to initialise ensemble currently this has a number of options:

11.14.1 Detailed Description

Definition at line 31 of file pf_control.f90.

11.14.2 Member Data Documentation

11.14.2.1 integer pf_control::pf_control_type::count

number of ensemble members associated with this MPI process

Definition at line 76 of file pf_control.f90.

11.14.2.2 integer pf_control::pf_control_type::couple_root

empire master processor

Definition at line 59 of file pf_control.f90.

11.14.2.3 real(kind=kind(1.0d0)) pf_control::pf_control_type::efac

Definition at line 45 of file pf_control.f90.

11.14.2.4 character(2) pf_control::pf_control_type::filter

which filter to use currently this has a number of options:

- SE – a stochastic ensemble
- DE – a deterministic ensemble
- SI – the SIR filter
- LE – the L-ETKF with noise
- LD – the L-ETKF without noise
- EW – the Equivalent Weights filter

- EZ – the Zhu equal weights filter particle filter
- LS – the L-ETKS with noise
- 3D – 3DVar

  Todo change these to a longer string

Definition at line 78 of file pf_control.f90.

11.14.2.5 logical pf_control::pf_control_type::gen_data

true generates synthetic obs for a twin experiment

Definition at line 37 of file pf_control.f90.

11.14.2.6 logical pf_control::pf_control_type::gen_q

true attempts to build up Q from long model run. UNUSED. DOES NOTHING!

Definition at line 38 of file pf_control.f90.

11.14.2.7 character(1) pf_control::pf_control_type::init

which method to initialise ensemble currently this has a number of options:

- N – perturb around the model initial conditions with random noise distributed $\mathcal{N}(0, I)$
- P – perturb around the model initial conditions with random noise distributed $\mathcal{N}(0, Q)$
- B – perturb around the model initial conditions with random noise distributed $\mathcal{N}(0, B)$
- R – read model states from rstrt folder where each ensemble member is stored in the file rstrt/##.state
- S – read model states from start folder where each ensemble member is stored in the file start/##.state
- U – call user defined perturbation routine. This assumes the user has implemented their own perturbation in user_perturb_particle
- Z – do not perturb particles. This will assume each model is received with initial spread

Definition at line 92 of file pf_control.f90.

11.14.2.8 real(kind=kind(1.0d0)) pf_control::pf_control_type::keep

proportion of particles to keep in EWPF EW step

Definition at line 46 of file pf_control.f90.

11.14.2.9 real(kind=kind(1.0d0)) pf_control::pf_control_type::len

R localisation length scale The entries in the observation error covariance matrix R are multiplied by the function $\exp\left(\frac{\text{dist}^2}{2\text{len}^2}\right)$.

Definition at line 53 of file pf_control.f90.

11.14.2.10 real(kind=kind(1.0d0)), dimension(:), allocatable pf_control::pf_control_type::mean

mean state vector

Definition at line 42 of file pf_control.f90.

11.14.2.11 integer pf_control::pf_control_type::nens

the total number of ensemble members

Definition at line 32 of file pf_control.f90.

11.14.2.12 real(kind=kind(1.0d0)) pf_control::pf_control_type::nfac

standard deviation of normal distribution in mixture density

Definition at line 43 of file pf_control.f90.

11.14.2.13 real(kind=kind(1.0d0)) pf_control::pf_control_type::nudgefac

the nudging factor

Definition at line 36 of file pf_control.f90.

11.14.2.14 logical pf_control::pf_control_type::output_weights

switch if true outputs ensemble weights

Definition at line 63 of file pf_control.f90.

11.14.2.15 integer, dimension(:), allocatable pf_control::pf_control_type::particles

particles associates with this MPI process

Definition at line 77 of file pf_control.f90.

11.14.2.16 real(kind=kind(1.0d0)), dimension(:,:), allocatable pf_control::pf_control_type::psi

state vector of ensemble members on this mpi process

Definition at line 41 of file pf_control.f90.

11.14.2.17 real(kind=kind(1.0d0)) pf_control::pf_control_type::qscale

scalar to multiply Q by

Definition at line 48 of file pf_control.f90.

11.14.2.18 real(kind=kind(1.0d0)) pf_control::pf_control_type::rho

enkf inflation factor so that $P_f = (1+\rho)^2 P_f$

Definition at line 50 of file pf_control.f90.

11.14.2.19 character(250) pf_control::pf_control_type::rmse_filename

string to hold the name of the file to output rmse to

Definition at line 72 of file pf_control.f90.

11.14.2.20 integer, dimension(:,:), allocatable pf_control::pf_control_type::talagrand

storage for rank histograms

Definition at line 75 of file pf_control.f90.

11.14.2.21 real(kind=kind(1.0d0)) pf_control::pf_control_type::time

dunno

Definition at line 47 of file pf_control.f90.

11.14.2.22 integer pf_control::pf_control_type::time_bwn_obs

the number of model timesteps between observations

Definition at line 35 of file pf_control.f90.

11.14.2.23 integer pf_control::pf_control_type::time_obs

the number of observations we will assimilate

Definition at line 34 of file pf_control.f90.

11.14.2.24 integer pf_control::pf_control_type::timestep =0

the current timestep as the model progresses

Definition at line 40 of file pf_control.f90.

11.14.2.25 real(kind=kind(1.0d0)) pf_control::pf_control_type::ufac

half width of the uniform distribution in mixture density

Definition at line 44 of file pf_control.f90.

11.14.2.26 logical pf_control::pf_control_type::use_ens_rmse

switich if true outputs the field of root mean squared errors where $rmse(j) = \sqrt{\frac{1}{N_e}\sum_{i=1}^{N_e}(x_i(j) - x^t(j))^2}$

Definition at line 69 of file pf_control.f90.

11.14.2.27 logical pf_control::pf_control_type::use_mean

switch if true outputs ensemble mean

Definition at line 64 of file pf_control.f90.

11.14.2.28 logical pf_control::pf_control_type::use_spatial_rmse

switch if true outputs Root Mean Square Errors
See Outputting Root Mean Squared Errors for more information

Definition at line 67 of file pf_control.f90.

11.14.2.29 logical pf_control::pf_control_type::use_talagrand

switch if true outputs rank histograms. See load_histogram_data for details.

Definition at line 60 of file pf_control.f90.

11.14.2.30 logical pf_control::pf_control_type::use_traj

switch if true outputs trajectories

Definition at line 66 of file pf_control.f90.

11.14.2.31 logical pf_control::pf_control_type::use_variance

switch if true outputs ensemble variance

Definition at line 65 of file pf_control.f90.

11.14.2.32 real(kind=kind(1.0d0)), dimension(:), allocatable pf_control::pf_control_type::weight

the negative log of the weights of the particles

Definition at line 33 of file pf_control.f90.

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

- src/controllers/pf_control.f90

11.15 qdata Module Reference

Module as a place to store user specified data for Q.

Public Member Functions

- subroutine loadq

  Subroutine to load in user data for Q.
- subroutine killq

  SUbroutine to deallocate user data for Q.

11.15.1 Detailed Description

Module as a place to store user specified data for Q.

- the model error covariance matrix

Definition at line 31 of file Qdata.f90.

11.15.2 Member Function/Subroutine Documentation

11.15.2.1 subroutine qdata::killq ( )

SUbroutine to deallocate user data for Q.

Definition at line 39 of file Qdata.f90.

11.15.2.2 subroutine qdata::loadq ( )

Subroutine to load in user data for Q.

Definition at line 35 of file Qdata.f90.

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

- src/user/Qdata.f90

11.16 random Module Reference

A module for random number generation from the following distributions:

Public Member Functions

- real(kind=kind(1.0d+0)) function random_normal ()

  function to get random normal with zero mean and stdev 1
- real(kind=kind(1.0d+0)) function random_gamma (s, first)
- real(kind=kind(1.0d+0)) function random_gamma1 (s, first)
- real(kind=kind(1.0d+0)) function random_gamma2 (s, first)
- real(kind=kind(1.0d+0)) function random_chisq (ndf, first)
- real(kind=kind(1.0d+0)) function random_exponential ()
- real(kind=kind(1.0d+0)) function random_weibull (a)
- real(kind=kind(1.0d+0)) function random_beta (aa, bb, first)
- real(kind=kind(1.0d+0)) function random_t (m)
- subroutine random_mvnorm (n, h, d, f, first, x, ier)
- real(kind=kind(1.0d+0)) function random_inv_gauss (h, b, first)
- integer function random_poisson (mu, first)
- integer function random_binomial1 (n, p, first)
- real(kind=kind(1.0d+0)) function bin_prob (n, p, r)
- real(dp) function lngamma (x)
- integer function random_binomial2 (n, pp, first)
- integer function random_neg_binomial (sk, p)
- real(kind=kind(1.0d+0)) function random_von_mises (k, first)
- real(kind=kind(1.0d+0)) function random_cauchy ()
- subroutine random_order (order, n)
- subroutine seed_random_number (iounit)

Public Attributes

- integer, parameter dp = SELECTED_REAL_KIND(12, 60)

11.16.1 Detailed Description

A module for random number generation from the following distributions:

Distribution Function/subroutine name

Normal (Gaussian) random_normal Gamma random_gamma Chi-squared random_chisq Exponential random_↩ exponential Weibull random_Weibull Beta random_beta t random_t Multivariate normal random_mvnorm Generalized inverse Gaussian random_inv_gauss Poisson random_Poisson Binomial random_binomial1 ∗ random_↩ binomial2 ∗ Negative binomial random_neg_binomial von Mises random_von_Mises Cauchy random_Cauchy

Definition at line 22 of file random_d.f90.

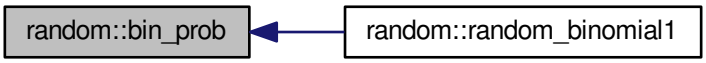
11.16.2 Member Function/Subroutine Documentation

11.16.2.1 real(kind=kind(1.0d+0)) function random::bin_prob (integer, intent(in) *n,* real(kind=kind(1.0d+0)), intent(in) *p,* integer, intent(in) *r*)

Definition at line 1000 of file random_d.f90.

Here is the call graph for this function:

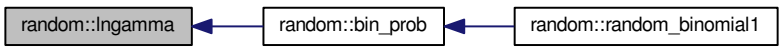
Here is the caller graph for this function:



11.16.2.2 real (dp) function random::lngamma (real (dp), intent(in) *x*)

Definition at line 1018 of file random_d.f90.

Here is the caller graph for this function:



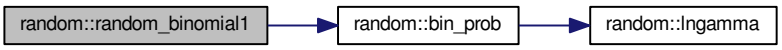
11.16.2.3 real(kind=kind(1.0d+0)) function random::random_beta (real(kind=kind(1.0d+0)), intent(in) *aa,* real(kind=kind(1.0d+0)), intent(in) *bb,* logical, intent(in) *first*)

Definition at line 371 of file random_d.f90.

11.16.2.4 integer function random::random_binomial1 (integer, intent(in) *n,* real(kind=kind(1.0d+0)), intent(in) *p,* logical, intent(in) *first*)

Definition at line 923 of file random_d.f90.

Here is the call graph for this function:



11.16.2.5 integer function random::random_binomial2 (integer, intent(in) *n,* real(kind=kind(1.0d+0)), intent(in) *pp,* logical, intent(in) *first*)

Definition at line 1082 of file random_d.f90.

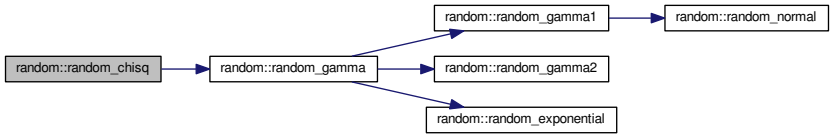
11.16.2.6 real(kind=kind(1.0d+0)) function random::random_cauchy ( )

Definition at line 1517 of file random_d.f90.

11.16.2.7 real(kind=kind(1.0d+0)) function random::random_chisq (integer, intent(in) *ndf,* logical, intent(in) *first*)

Definition at line 308 of file random_d.f90.

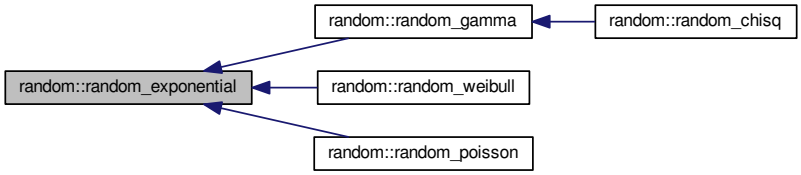
Here is the call graph for this function:



11.16.2.8 real(kind=kind(1.0d+0)) function random::random_exponential ( )

Definition at line 324 of file random_d.f90.

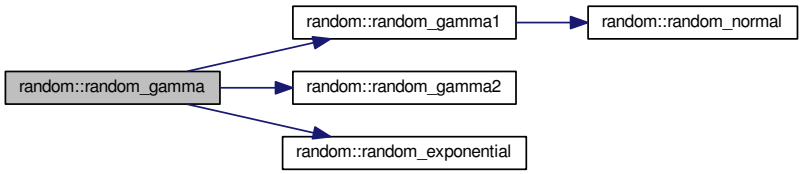
Here is the caller graph for this function:



11.16.2.9 real(kind=kind(1.0d+0)) function random::random_gamma (real(kind=kind(1.0d+0)), intent(in) *s,* logical, intent(in) *first*)

Definition at line 154 of file random_d.f90.

Here is the call graph for this function:



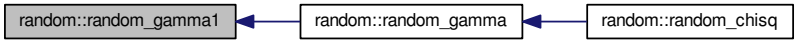
Here is the caller graph for this function:

11.16.2.10 real(kind=kind(1.0d+0)) function random::random_gamma1 (real(kind=kind(1.0d+0)), intent(in) *s,* logical, intent(in) *first*)

Definition at line 189 of file random_d.f90.

Here is the call graph for this function:

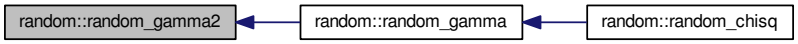
Here is the caller graph for this function:



11.16.2.11 real(kind=kind(1.0d+0)) function random::random_gamma2 (real(kind=kind(1.0d+0)), intent(in) *s,* logical, intent(in) *first*)

Definition at line 238 of file random_d.f90.

Here is the caller graph for this function:



11.16.2.12 real(kind=kind(1.0d+0)) function random::random_inv_gauss (real(kind=kind(1.0d+0)), intent(in) *h,* real(kind=kind(1.0d+0)), intent(in) *b,* logical, intent(in) *first*)

Definition at line 610 of file random_d.f90.

11.16.2.13 subroutine random::random_mvnorm (integer, intent(in) *n,* real(kind=kind(1.0d+0)), dimension(:), intent(in) *h,* real(kind=kind(1.0d+0)), dimension(:), intent(in) *d,* real(kind=kind(1.0d+0)), dimension(:), intent(inout) *f,* logical, intent(in) *first,* real(kind=kind(1.0d+0)), dimension(:), intent(out) *x,* integer, intent(out) *ier*)

Definition at line 509 of file random_d.f90.

Here is the call graph for this function:

11.16.2.14 integer function random::random_neg_binomial (real(kind=kind(1.0d+0)), intent(in) *sk,* real(kind=kind(1.0d+0)), intent(in) *p*)

Definition at line 1314 of file random_d.f90.

11.16.2.15 real(kind=kind(1.0d+0)) function random::random_normal ( )

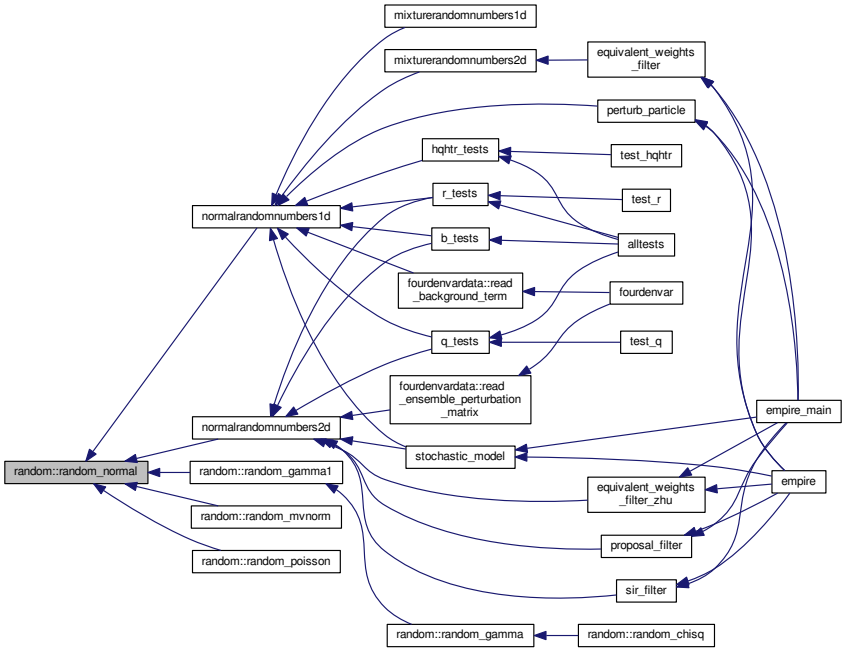
function to get random normal with zero mean and stdev 1

Returns

fn_val

Definition at line 108 of file random_d.f90.

Here is the caller graph for this function:



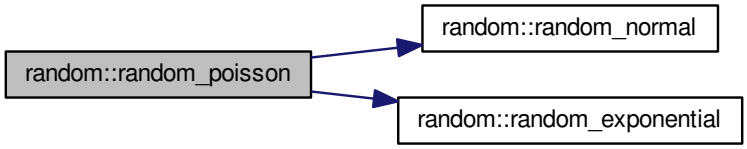
11.16.2.16 subroutine random::random_order (integer, dimension(n), intent(out) *order,* integer, intent(in) *n*)

Definition at line 1539 of file random_d.f90.

11.16.2.17 integer function random::random_poisson (real(kind=kind(1.0d+0)), intent(in) *mu,* logical, intent(in) *first*)

Definition at line 681 of file random_d.f90.

Here is the call graph for this function:



11.16.2.18 real(kind=kind(1.0d+0)) function random::random_t (integer, intent(in) *m*)

Definition at line 448 of file random_d.f90.

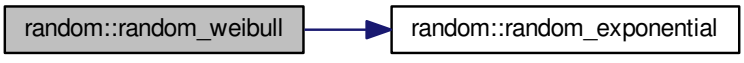
11.16.2.19 real(kind=kind(1.0d+0)) function random::random_von_mises (real(kind=kind(1.0d+0)), intent(in) *k,* logical, intent(in) *first*)

Definition at line 1389 of file random_d.f90.

11.16.2.20 real(kind=kind(1.0d+0)) function random::random_weibull (real(kind=kind(1.0d+0)), intent(in) *a*)

Definition at line 351 of file random_d.f90.

Here is the call graph for this function:



11.16.2.21 subroutine random::seed_random_number (integer, intent(in) *iounit*)

Definition at line 1573 of file random_d.f90.

11.16.3 Member Data Documentation

11.16.3.1 integer, parameter random::dp = SELECTED_REAL_KIND(12, 60)

Definition at line 101 of file random_d.f90.

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

- src/utils/random_d.f90

11.17 random_number_controls Module Reference

Public Member Functions

- subroutine set_random_number_controls

Public Attributes

- character(10) normal_generator ='random_d'

11.17.1 Detailed Description

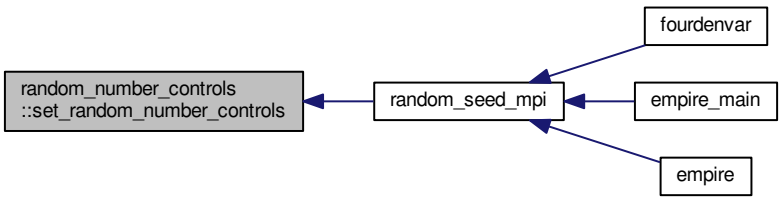
Definition at line 27 of file gen_rand.f90.

11.17.2 Member Function/Subroutine Documentation

11.17.2.1 subroutine random_number_controls::set_random_number_controls ( )

Definition at line 30 of file gen_rand.f90.

Here is the caller graph for this function:



11.17.3 Member Data Documentation

11.17.3.1 character(10) random_number_controls::normal_generator ='random_d'

Definition at line 28 of file gen_rand.f90.

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

- src/operations/gen_rand.f90

11.18 rdata Module Reference

Module to hold user supplied data for R observation error covariance matrix.

Public Member Functions

- subroutine loadr

  Subroutine to load data for R.
- subroutine killr

  SUbroutine to deallocate R data.

11.18.1 Detailed Description

Module to hold user supplied data for R observation error covariance matrix.

Definition at line 31 of file Rdata.f90.

11.18.2 Member Function/Subroutine Documentation

11.18.2.1 subroutine rdata::killr ( )

SUbroutine to deallocate R data.

Definition at line 39 of file Rdata.f90.

11.18.2.2 subroutine rdata::loadr ( )

Subroutine to load data for R.

Definition at line 35 of file Rdata.f90.

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

- src/user/Rdata.f90

11.19 sizes Module Reference

Module that stores the dimension of observation and state spaces.

Public Attributes

- integer obs_dim

  size of the observations held on this process. For empire versions 1 and 2, this is the total number of observations
- integer state_dim

  size of the state held on this process. For empire versions 1 and 2, this is the total size of the state vector
- integer obs_dim_g

  global size of obs dim over all processes
- integer state_dim_g

  global size of state dim over all processes

11.19.1 Detailed Description

Module that stores the dimension of observation and state spaces.

Definition at line 29 of file sizes.f90.

11.19.2 Member Data Documentation

11.19.2.1 integer sizes::obs_dim

size of the observations held on this process. For empire versions 1 and 2, this is the total number of observations

Definition at line 31 of file sizes.f90.

11.19.2.2 integer sizes::obs_dim_g

global size of obs dim over all processes

Definition at line 39 of file sizes.f90.

11.19.2.3 integer sizes::state_dim

size of the state held on this process. For empire versions 1 and 2, this is the total size of the state vector

Definition at line 35 of file sizes.f90.

11.19.2.4 integer sizes::state_dim_g

global size of state dim over all processes

Definition at line 40 of file sizes.f90.

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

- src/controllers/sizes.f90

11.20 threedvar_data Module Reference

module to store stuff for 3DVar

Public Attributes

- real(kind=kind(1.0d0)), dimension(:), allocatable xb

  the background guess

11.20.1 Detailed Description

module to store stuff for 3DVar

Definition at line 29 of file threedvar_data.f90.

11.20.2 Member Data Documentation

11.20.2.1 real(kind=kind(1.0d0)), dimension(:), allocatable threedvar_data::xb

the background guess

Definition at line 31 of file threedvar_data.f90.

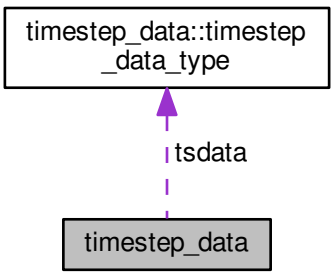
The documentation for this module was generated from the following file:

- src/var/threedvar_data.f90

11.21 timestep_data Module Reference

Module that stores the information about the timestepping process.

Collaboration diagram for timestep_data:



Data Types

- type timestep_data_type

Public Member Functions

- subroutine timestep_data_allocate_obs_times (n)

  subroutine to allocate space for obs_times array
- subroutine timestep_data_deallocate_obs_times

  subroutine to deallocate obs_times array
- subroutine timestep_data_set_obs_times (obs_num_in_time, timestep)

  subroutine to set the timestep corresponding to the observation number in time
- subroutine timestep_data_set_next_ob_time (ob_time)

  subroutine to set the next observation timestep
- subroutine timestep_data_get_obs_times (obs_num_in_time, timestep)

  subroutine to extract the timestep corresponding to the observation number in time
- subroutine timestep_data_set_do_analysis

  subroutine to define if the current timestep should perform an analysis
- subroutine timestep_data_set_do_no_analysis

subroutine to define if the current timestep should not perform an analysis

- subroutine timestep_data_set_is_analysis

  subroutine to define if the current ensemble is an analysis
- subroutine timestep_data_set_no_analysis

  subroutine to define if the current ensemble is not an analysis
- subroutine timestep_data_set_completed (t)

  subroutine to define the number of completed timesteps
- subroutine timestep_data_set_current (t)

  subroutine to define the current timestep
- subroutine timestep_data_set_total (t)

  subroutine to define the total number of timesteps that the model will run for
- subroutine timestep_data_set_tau (pseudotimestep)

  subroutine to define the current number of timesteps between observations

Public Attributes

- type(timestep_data_type), save tsdata

  the derived data type holding all timestep data

11.21.1 Detailed Description

Module that stores the information about the timestepping process.

Definition at line 30 of file timestep_data.f90.

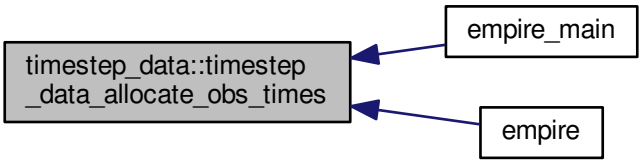
11.21.2 Member Function/Subroutine Documentation

11.21.2.1 subroutine timestep_data::timestep_data_allocate_obs_times (integer, intent(in) *n*)

subroutine to allocate space for obs_times array

Definition at line 60 of file timestep_data.f90.

Here is the caller graph for this function:



11.21.2.2 subroutine timestep_data::timestep_data_deallocate_obs_times ( )

subroutine to deallocate obs_times array

Definition at line 74 of file timestep_data.f90.

11.21.2.3 subroutine timestep_data::timestep_data_get_obs_times (integer, intent(in) *obs_num_in_time,* integer, intent(out) *timestep*)

subroutine to extract the timestep corresponding to the observation number in time

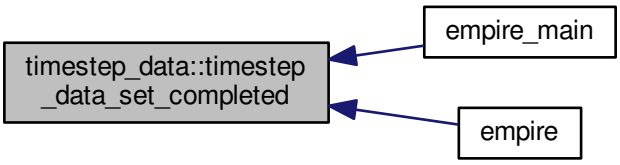
Definition at line 98 of file timestep_data.f90.

11.21.2.4 subroutine timestep_data::timestep_data_set_completed (integer, intent(in) *t*)

subroutine to define the number of completed timesteps

Definition at line 131 of file timestep_data.f90.

Here is the caller graph for this function:

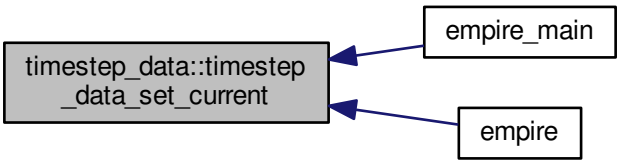


11.21.2.5 subroutine timestep_data::timestep_data_set_current (integer, intent(in) *t*)

subroutine to define the current timestep

Definition at line 138 of file timestep_data.f90.

Here is the caller graph for this function:

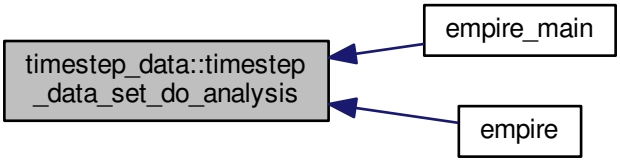


11.21.2.6 subroutine timestep_data::timestep_data_set_do_analysis ( )

subroutine to define if the current timestep should perform an analysis

Definition at line 106 of file timestep_data.f90.

Here is the caller graph for this function:

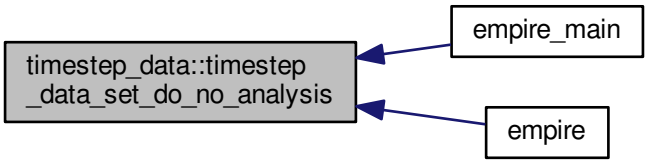


11.21.2.7 subroutine timestep_data::timestep_data_set_do_no_analysis ( )

subroutine to define if the current timestep should not perform an analysis

Definition at line 113 of file timestep_data.f90.

Here is the caller graph for this function:

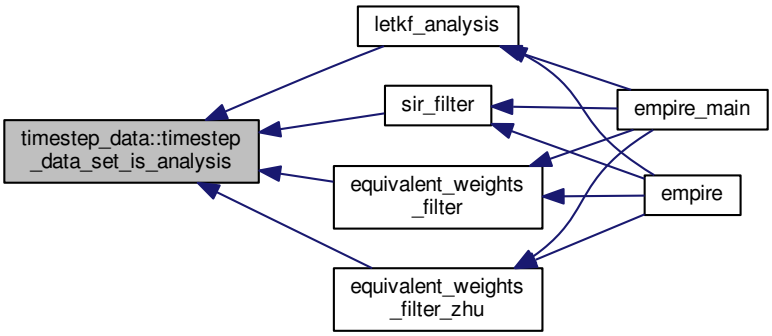


11.21.2.8 subroutine timestep_data::timestep_data_set_is_analysis ( )

subroutine to define if the current ensemble is an analysis

Definition at line 119 of file timestep_data.f90.

Here is the caller graph for this function:

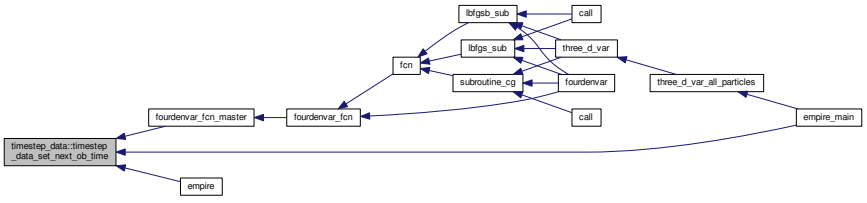


11.21.2.9 subroutine timestep_data::timestep_data_set_next_ob_time (integer, intent(in) *ob_time*)

subroutine to set the next observation timestep

Definition at line 89 of file timestep_data.f90.

Here is the caller graph for this function:

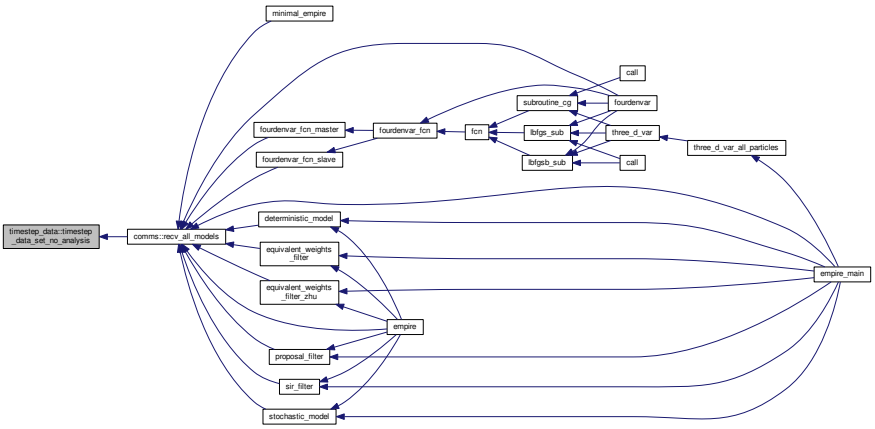


11.21.2.10 subroutine timestep_data::timestep_data_set_no_analysis ( )

subroutine to define if the current ensemble is not an analysis

Definition at line 125 of file timestep_data.f90.

Here is the caller graph for this function:

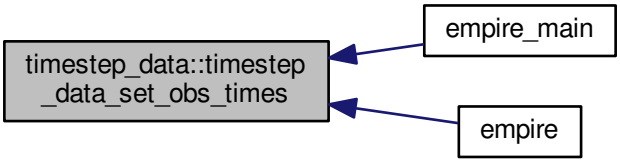


11.21.2.11 subroutine timestep_data::timestep_data_set_obs_times (integer, intent(in) *obs_num_in_time,* integer, intent(in) *timestep*)

subroutine to set the timestep corresponding to the observation number in time

Definition at line 80 of file timestep_data.f90.

Here is the caller graph for this function:

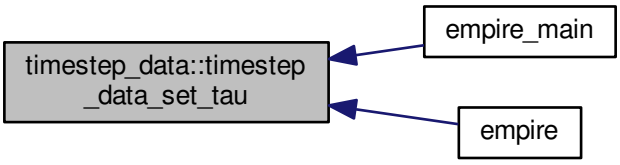


11.21.2.12 subroutine timestep_data::timestep_data_set_tau (integer, intent(in) *pseudotimestep*)

subroutine to define the current number of timesteps between observations

Definition at line 153 of file timestep_data.f90.

Here is the caller graph for this function:

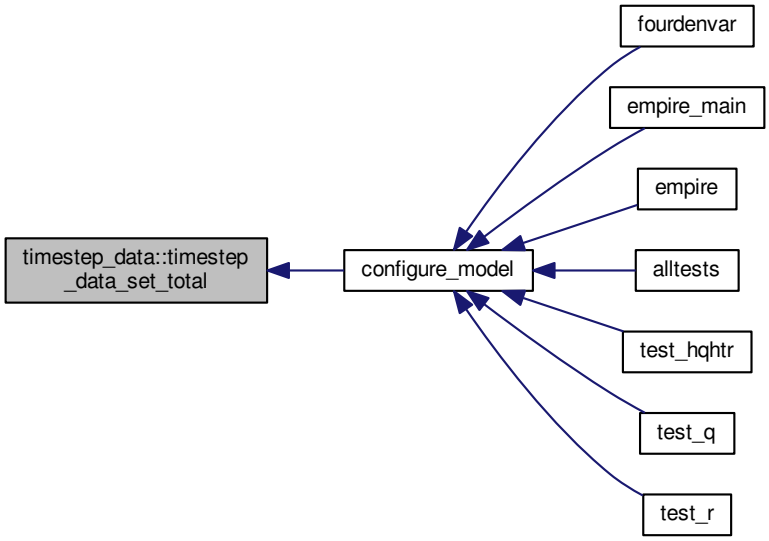


11.21.2.13 subroutine timestep_data::timestep_data_set_total (integer, intent(in) *t*)

subroutine to define the total number of timesteps that the model will run for

Definition at line 146 of file timestep_data.f90.

Here is the caller graph for this function:



11.21.3 Member Data Documentation

11.21.3.1 type(timestep_data_type), save timestep_data::tsdata

the derived data type holding all timestep data

Definition at line 55 of file timestep_data.f90.

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

- src/controllers/timestep_data.f90

11.22 timestep_data::timestep_data_type Type Reference

Public Attributes

- integer total_timesteps

  total number of timesteps that the model will run
- integer current_timestep

  the current timestep that empire is running
- integer completed_timesteps

  the number of timesteps that empire has so far finished
- integer next_ob_timestep

  the timestep of the next observation
- logical is_analysis

  if true, then the current ensemble is an analysis. If false then the current ensemble is not an analysis
- logical do_analysis

  if true then on this timestep we are required to do an analysis. If false we do not have an observation at this timestep
- integer, dimension(:), allocatable obs_times

  an integer array that will hold a mapping from observation number in time to model timesteps. I.e. obs_times(i) is the timestep of observation i in time.
- integer tau

  the pseudotimestep between observations

11.22.1 Detailed Description

Definition at line 32 of file timestep_data.f90.

11.22.2 Member Data Documentation

11.22.2.1 integer timestep_data::timestep_data_type::completed_timesteps

the number of timesteps that empire has so far finished

Definition at line 37 of file timestep_data.f90.

11.22.2.2 integer timestep_data::timestep_data_type::current_timestep

the current timestep that empire is running

Definition at line 35 of file timestep_data.f90.

11.22.2.3 logical timestep_data::timestep_data_type::do_analysis

if true then on this timestep we are required to do an analysis. If false we do not have an observation at this timestep

Definition at line 43 of file timestep_data.f90.

11.22.2.4 logical timestep_data::timestep_data_type::is_analysis

if true, then the current ensemble is an analysis. If false then the current ensemble is not an analysis

Definition at line 40 of file timestep_data.f90.

11.22.2.5 integer timestep_data::timestep_data_type::next_ob_timestep

the timestep of the next observation

Definition at line 39 of file timestep_data.f90.

11.22.2.6 integer, dimension(:), allocatable timestep_data::timestep_data_type::obs_times

an integer array that will hold a mapping from observation number in time to model timesteps. I.e. obs_times(i) is the timestep of observation i in time.

Definition at line 47 of file timestep_data.f90.

11.22.2.7 integer timestep_data::timestep_data_type::tau

the pseudotimestep between observations

Definition at line 53 of file timestep_data.f90.

11.22.2.8 integer timestep_data::timestep_data_type::total_timesteps

total number of timesteps that the model will run

Definition at line 33 of file timestep_data.f90.

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

- src/controllers/timestep_data.f90

11.23 traj_data Module Reference

module to hold data for trajectories

Public Member Functions

- subroutine setup_traj

  subroutine to read in which trajectories are required
- subroutine deallocate_traj

Public Attributes

- integer trajn
- integer, dimension(:), allocatable trajvar
- character(28), parameter traj_list ='traj_list.dat'

11.23.1 Detailed Description

module to hold data for trajectories

Definition at line 28 of file trajectories.f90.

11.23.2 Member Function/Subroutine Documentation

11.23.2.1 subroutine traj_data::deallocate_traj ( )

Definition at line 137 of file trajectories.f90.

11.23.2.2 subroutine traj_data::setup_traj ( )

subroutine to read in which trajectories are required

this requires that the directory traj/ exists before runtime.

Then this reads the file traj_list .

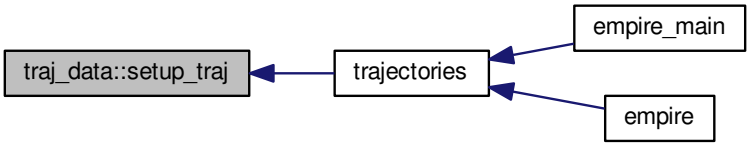
The format for traj_list is a list of K+1 integers,

where the first integer is K

and the following K integers are the index in the state dimension for which the trajectories are required.

Definition at line 45 of file trajectories.f90.

Here is the caller graph for this function:



11.23.3 Member Data Documentation

11.23.3.1 character(28), parameter traj_data::traj_list ='traj_list.dat'

Definition at line 31 of file trajectories.f90.

11.23.3.2 integer traj_data::trajn

Definition at line 29 of file trajectories.f90.

11.23.3.3 integer, dimension(:), allocatable traj_data::trajvar

Definition at line 30 of file trajectories.f90.

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

- src/utils/trajectories.f90

11.24 var_data::var_control_type Type Reference

Public Attributes

- character(6) opt_method

  which optimization method to use currently this has a number of options:

- integer cg_method

  which type of nonlinear CG method to use options are:
  1 FLETCHER-REEVES
  2 POLAK-RIBIERE (DEFAULT)
  3 POSITIVE POLAK-RIBIERE (BETA=MAX{BETA,0})
- real(kind=kind(1.0d0)) cg_eps

  convergence tolerance for CG method
  DEFAULT = 1.0d-5
- real(kind=kind(1.0d0)) lbfgs_factr

  factr is a DOUBLE PRECISION variable that must be set by the user.
  It is a tolerance in the termination test for the algorithm. The iteration will stop when
- real(kind=kind(1.0d0)) lbfgs_pgtol

  pgtol is a double precision variable.
  On entry pgtol >= 0 is specified by the user. The iteration will stop when
- real(kind=kind(1.0d0)), dimension(:), allocatable l
- real(kind=kind(1.0d0)), dimension(:), allocatable u
- real(kind=kind(1.0d0)), dimension(:), allocatable x0
- integer, dimension(:), allocatable nbd
- integer n

  the size of the state vector
- integer total_timesteps

  the total number of timesteps in the assimilation window
- integer, dimension(:), allocatable ny

  array containing the number of observations.

11.24.1 Detailed Description

Definition at line 32 of file var_data.f90.

11.24.2 Member Data Documentation

11.24.2.1 real(kind=kind(1.0d0)) var_data::var_control_type::cg_eps

convergence tolerance for CG method
DEFAULT = 1.0d-5

Definition at line 44 of file var_data.f90.

11.24.2.2 integer var_data::var_control_type::cg_method

which type of nonlinear CG method to use options are:
1 FLETCHER-REEVES
2 POLAK-RIBIERE (DEFAULT)
3 POSITIVE POLAK-RIBIERE (BETA=MAX{BETA,0})

Definition at line 39 of file var_data.f90.

11.24.2.3 real(kind=kind(1.0d0)), dimension(:), allocatable var_data::var_control_type::l

Definition at line 78 of file var_data.f90.

11.24.2.4 real(kind=kind(1.0d0)) var_data::var_control_type::lbfgs_factr

factr is a DOUBLE PRECISION variable that must be set by the user.
It is a tolerance in the termination test for the algorithm. The iteration will stop when

(f^k - f^{k+1})/max{|f^k|,|f^{k+1}|,1} <= factr∗epsmch

where epsmch is the machine precision which is automatically generated by the code. Typical values for factr on a computer with 15 digits of accuracy in double precision are:
factr=1.d+12 for low accuracy;
1.d+7 for moderate accuracy;
1.d+1 for extremely high accuracy.
The user can suppress this termination test by setting factr=0.

DEFAULT = 1.0d7

Definition at line 48 of file var_data.f90.

11.24.2.5 real(kind=kind(1.0d0)) var_data::var_control_type::lbfgs_pgtol

pgtol is a double precision variable.
On entry pgtol >= 0 is specified by the user. The iteration will stop when

max{|proj g_i | i = 1, ..., n} <= pgtol

where pg_i is the ith component of the projected gradient.
The user can suppress this termination test by setting pgtol=0.

DEFAULT = 1.0d-5

Definition at line 66 of file var_data.f90.

11.24.2.6 integer var_data::var_control_type::n

the size of the state vector

Definition at line 81 of file var_data.f90.

11.24.2.7 integer, dimension(:), allocatable var_data::var_control_type::nbd

Definition at line 79 of file var_data.f90.

11.24.2.8 integer, dimension(:), allocatable var_data::var_control_type::ny

array containing the number of observations.

ny(t) contains the number of observations at time t

if no observations at time t then ny(t) = 0

Definition at line 85 of file var_data.f90.

11.24.2.9 character(6) var_data::var_control_type::opt_method

which optimization method to use currently this has a number of options:

- 'cg'

- 'lbfgs'
- 'lbfgsb'

Definition at line 33 of file var_data.f90.

11.24.2.10 integer var_data::var_control_type::total_timesteps

the total number of timesteps in the assimilation window

Definition at line 82 of file var_data.f90.

11.24.2.11 real(kind=kind(1.0d0)), dimension(:), allocatable var_data::var_control_type::u

Definition at line 78 of file var_data.f90.

11.24.2.12 real(kind=kind(1.0d0)), dimension(:), allocatable var_data::var_control_type::x0

Definition at line 78 of file var_data.f90.

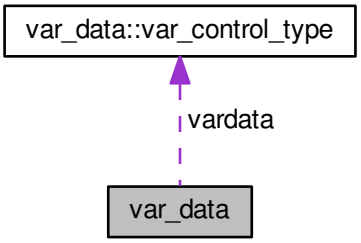
The documentation for this type was generated from the following file:

- src/4dEnVar/var_data.f90

11.25 var_data Module Reference

module holding data for variational problems

Collaboration diagram for var_data:



Data Types

- type var_control_type

Public Member Functions

- subroutine set_var_controls

  subroutine to ensure vardata is ok
- subroutine parse_vardata

  subroutine to read the namelist file and save it to vardata datatype Here we read vardata.nml
- subroutine allocate_vardata

  subroutine to allocate space for 4denvar
- subroutine deallocate_vardata

  subroutine to deallocate space for 4denvar
- subroutine read_lbfgsb_bounds

  subroutine to somehow read in bounds data
- subroutine read_observation_numbers

  subroutine to somehow read in observation numbers

Public Attributes

- type(var_control_type), save vardata

  the derived data type holding all controlling data

11.25.1 Detailed Description

module holding data for variational problems

Definition at line 29 of file var_data.f90.

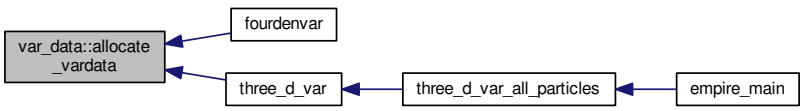
11.25.2 Member Function/Subroutine Documentation

11.25.2.1 subroutine var_data::allocate_vardata ( )

subroutine to allocate space for 4denvar

Definition at line 328 of file var_data.f90.

Here is the caller graph for this function:

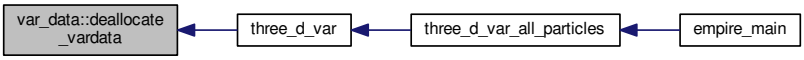


11.25.2.2 subroutine var_data::deallocate_vardata ( )

subroutine to deallocate space for 4denvar

Definition at line 343 of file var_data.f90.

Here is the caller graph for this function:



11.25.2.3 subroutine var_data::parse_vardata ( )

subroutine to read the namelist file and save it to vardata datatype Here we read vardata.nml

vardata.nml is a fortran namelist file. As such, within it there must be a line beginning

&var_params

To make it (probably) work, ensure there is a forward slash on the penultimate line and a blank line to end the file

This is just the fortran standard for namelists though.

On to the content...in any order, the vardata.nml may contain the following things:

Integers:

- cg_method

Reals, double precision:

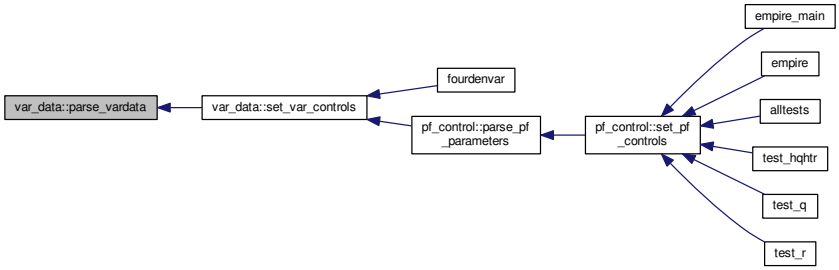
- lbfgs_factr
- lbfgs_pgtol

6 Characters:

- opt_method

Definition at line 139 of file var_data.f90.

Here is the caller graph for this function:

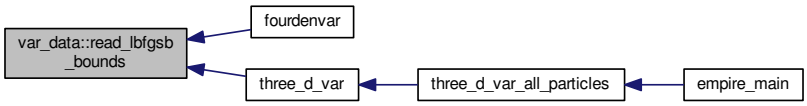


11.25.2.4 subroutine var_data::read_lbfgsb_bounds ( )

subroutine to somehow read in bounds data

Definition at line 353 of file var_data.f90.

Here is the caller graph for this function:

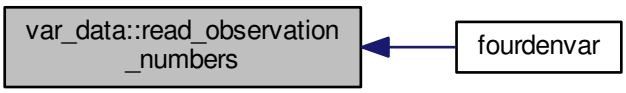


11.25.2.5 subroutine var_data::read_observation_numbers ( )

subroutine to somehow read in observation numbers

Definition at line 357 of file var_data.f90.

Here is the caller graph for this function:



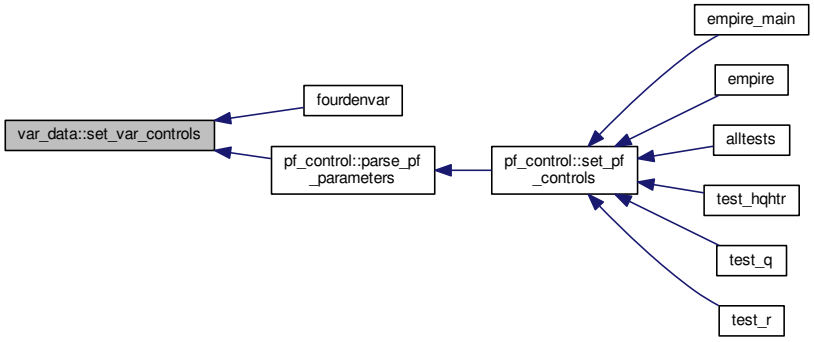
11.25.2.6 subroutine var_data::set_var_controls ( )

subroutine to ensure vardata is ok

Definition at line 98 of file var_data.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



11.25.3 Member Data Documentation

11.25.3.1 type(var_control_type), save var_data::vardata

the derived data type holding all controlling data

Definition at line 93 of file var_data.f90.

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

- src/4dEnVar/var_data.f90

11.26 ziggurat Module Reference

Public Member Functions

- subroutine, public zigset (jsrseed)
- integer function, public shr3 ()
- real(dp) function, public uni ()
- real(dp) function, public rnor ()
- real(dp) function, public rexp ()

11.26.1 Detailed Description

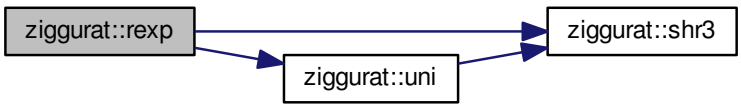
Definition at line 17 of file ziggurat.f90.

11.26.2 Member Function/Subroutine Documentation

11.26.2.1 real(dp) function, public ziggurat::rexp ( )

Definition at line 202 of file ziggurat.f90.

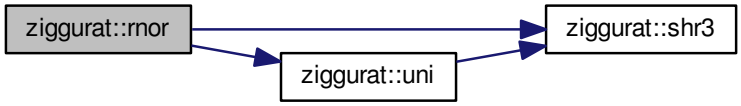
Here is the call graph for this function:



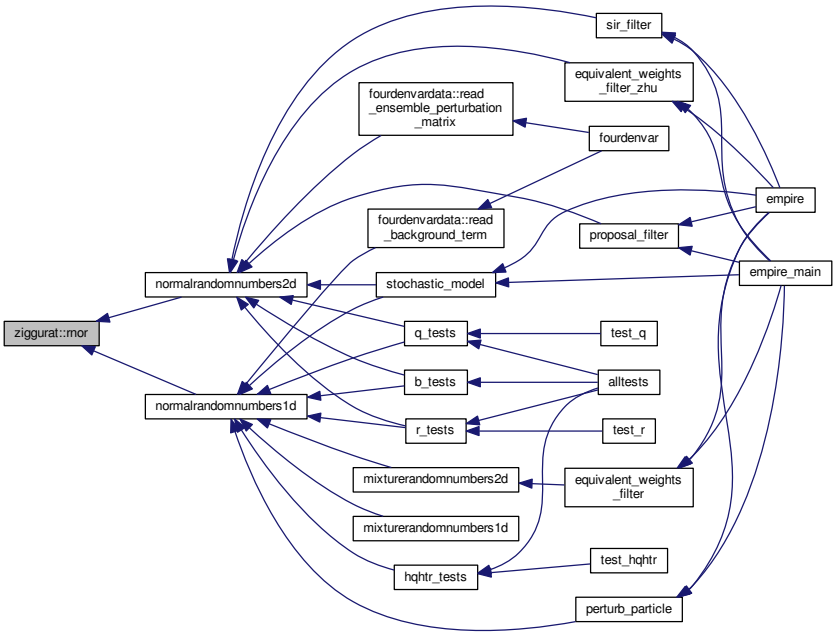
11.26.2.2 real(dp) function, public ziggurat::rnor ( )

Definition at line 161 of file ziggurat.f90.

Here is the call graph for this function:



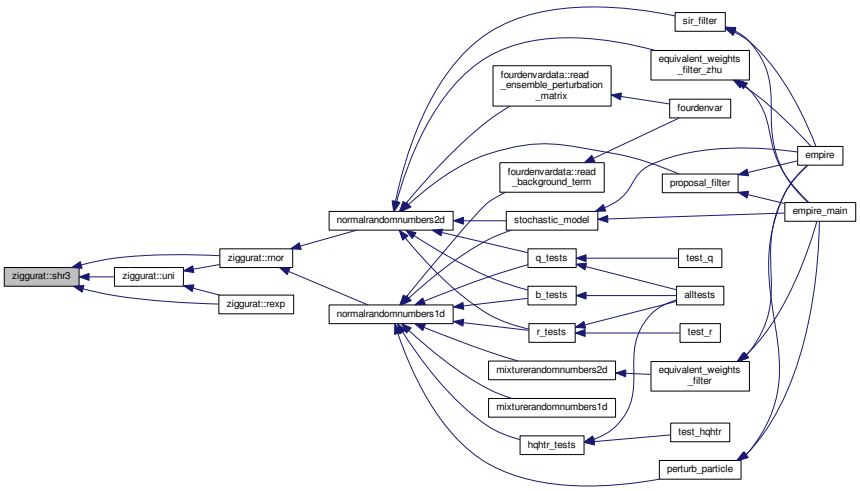
Here is the caller graph for this function:



11.26.2.3 integer function, public ziggurat::shr3 ( )

Definition at line 133 of file ziggurat.f90.

Here is the caller graph for this function:

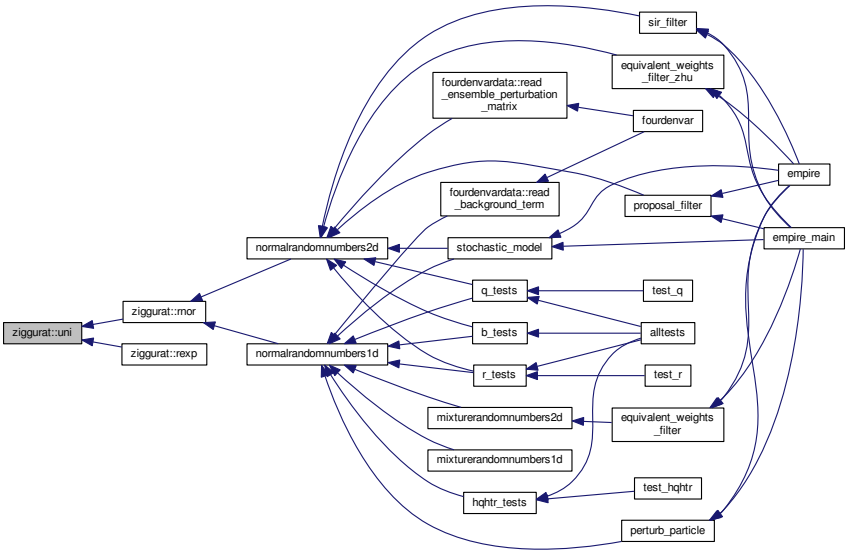


11.26.2.4 real(dp) function, public ziggurat::uni ( )

Definition at line 151 of file ziggurat.f90.

Here is the call graph for this function:

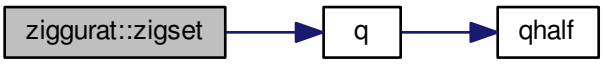
Here is the caller graph for this function:



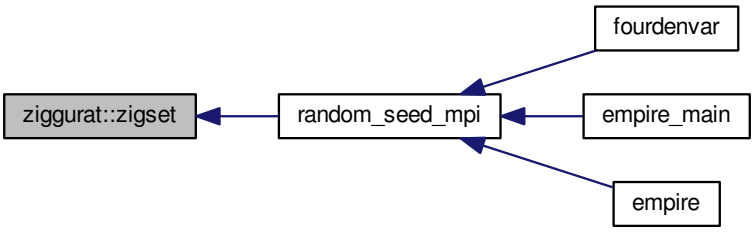
11.26.2.5 subroutine, public ziggurat::zigset (integer, intent(in) *jsrseed*)

Definition at line 64 of file ziggurat.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- src/utils/ziggurat.f90

Chapter 12

File Documentation

12.1 comm_version.f90 File Reference

Data Types

- module communicator_version

  module to store the parameter comm_version to control the communication pattern that empire will use.

12.2 doc/doxygen/cite.txt File Reference

12.3 doc/doxygen/empire_comms.txt File Reference

12.4 doc/doxygen/methods.txt File Reference

12.5 doc/doxygen/other_features.txt File Reference

12.6 doc/doxygen/tutorial_lorenz96.txt File Reference

12.7 doc/doxygen/tutorials.txt File Reference

12.8 model_specific.f90 File Reference

Functions/Subroutines

- subroutine configure_model

  subroutine called initially to set up details and data for model specific functions
- subroutine reconfigure_model

  subroutine to reset variables that may change when the observation network changes
- subroutine solve_r (obsDim, nrhs, y, v, t)

  subroutine to take an observation vector y and return v in observation space.
- subroutine solve_rhalf (obsdim, nrhs, y, v, t)

  subroutine to take an observation vector y and return v in observation space.
- subroutine solve_hqht_plus_r (obsdim, y, v, t)

  subroutine to take an observation vector y and return v in observation space.

- subroutine q (nrhs, x, Qx)

  subroutine to take a full state vector x and return Qx in state space.

- subroutine qhalf (nrhs, x, Qx)

  subroutine to take a full state vector x and return $Q^{1/2}x$ in state space.

- subroutine r (obsDim, nrhs, y, Ry, t)

  subroutine to take an observation vector x and return Rx in observation space.

- subroutine rhalf (obsDim, nrhs, y, Ry, t)

  subroutine to take an observation vector x and return Rx in observation space.

- subroutine h (obsDim, nrhs, x, hx, t)

  subroutine to take a full state vector x and return H(x) in observation space.

- subroutine ht (obsDim, nrhs, y, x, t)

  subroutine to take an observation vector y and return $x = H^T(y)$ in full state space.

- subroutine dist_st_ob (xp, yp, dis, t)

  subroutine to compute the distance between the variable in the state vector and the variable in the observations

- subroutine bhalf (nrhs, x, bx)

  subroutine to take a full state vector x and return $B^{1/2}x$ in state space.

- subroutine solve_b (nrhs, x, v)

  subroutine to take a state vector x and return v in state space.

- subroutine get_observation_data (y, t)

  Subroutine to read observation from a file
  .

12.8.1 Function/Subroutine Documentation

12.8.1.1 subroutine bhalf (integer, intent(in) *nrhs,* real(kind=rk), dimension(state_dim,nrhs), intent(in) *x,* real(kind=rk), dimension(state_dim,nrhs), intent(out) *bx*)

subroutine to take a full state vector x and return $B^{1/2}x$ in state space.

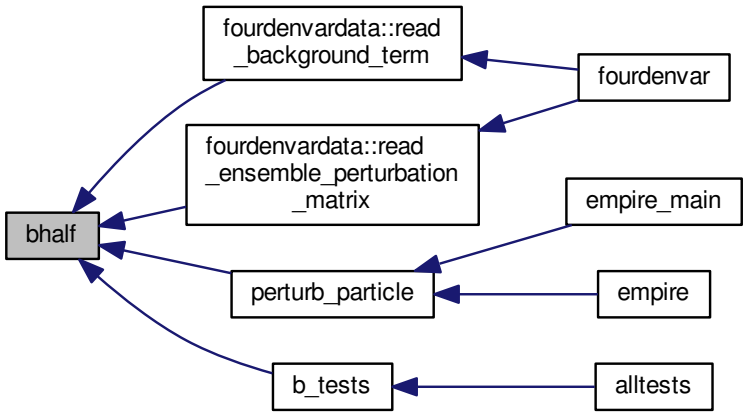
Given x compute $B^{\frac{1}{2}}x$

Parameters

in	*nrhs*	the number of right hand sides
in	*x*	the input vector
out	*bx*	the resulting vector where bx $= B^{\frac{1}{2}}x$

Definition at line 281 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.2 subroutine configure_model ( )

subroutine called initially to set up details and data for model specific functions

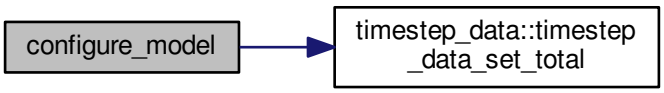
By the end of this subroutine, the following must be set:

- state_dim in sizes
- obs_dim in sizes for the first observation
- total_timesteps in timestep_data

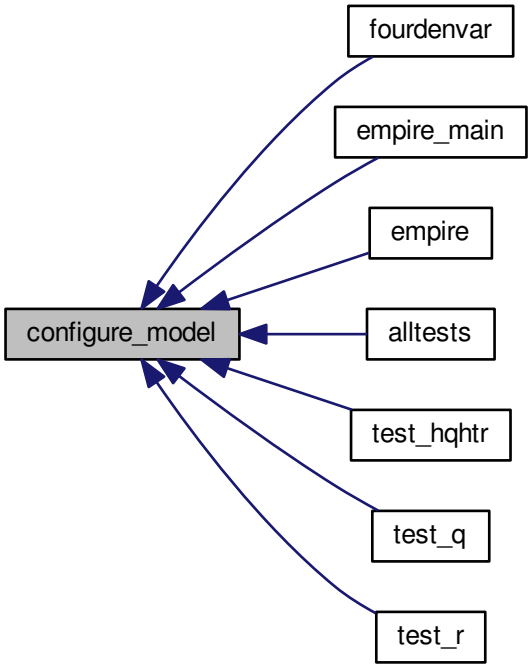
This is a very good place to load in data for the matrices B,Q,R,H etc

Definition at line 38 of file model_specific.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.8.1.3 subroutine dist_st_ob (integer, intent(in) *xp,* integer, intent(in) *yp,* real(kind=kind(1.0d0)), intent(out) *dis,* integer, intent(in) *t*)

subroutine to compute the distance between the variable in the state vector and the variable in the observations

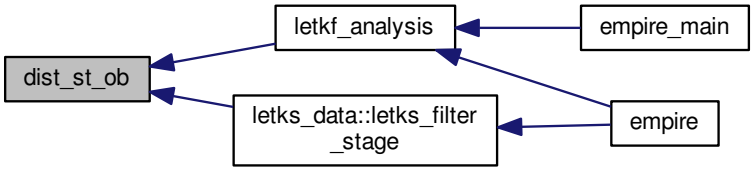
Compute $\mathrm{dist}(x(xp), y(yp))$

Parameters

in	*xp*	the index in the state vector
in	*yp*	the index in the observation vector
out	*dis*	the distance between x(xp) and y(yp)
in	*t*	the current time index for observations

Definition at line 265 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.4 subroutine get_observation_data (real(kind=rk), dimension(obs_dim), intent(out) *y*, integer, intent(in) *t*)

Subroutine to read observation from a file

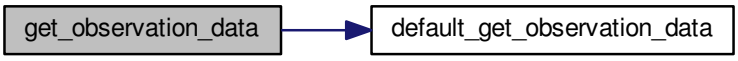
.

Parameters

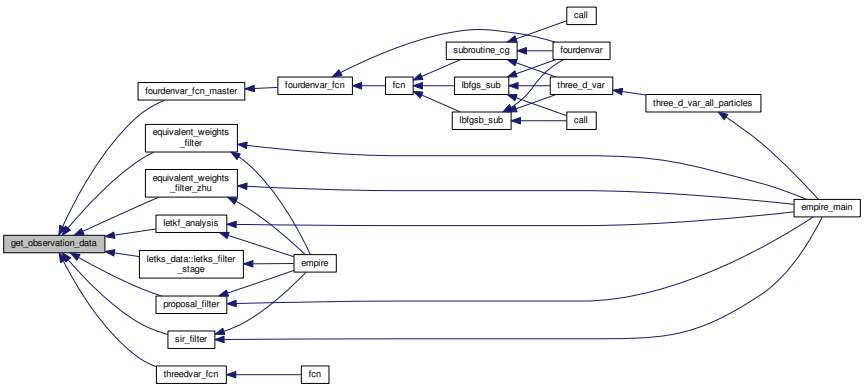
out	*y*	The observation
in	*t*	the current timestep

Definition at line 319 of file model_specific.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.8.1.5 subroutine h (integer, intent(in) *obsDim,* integer, intent(in) *nrhs,* real(kind=rk), dimension(state_dim,nrhs), intent(in) *x,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *hx,* integer, intent(in) *t*)

subroutine to take a full state vector x and return H(x) in observation space.

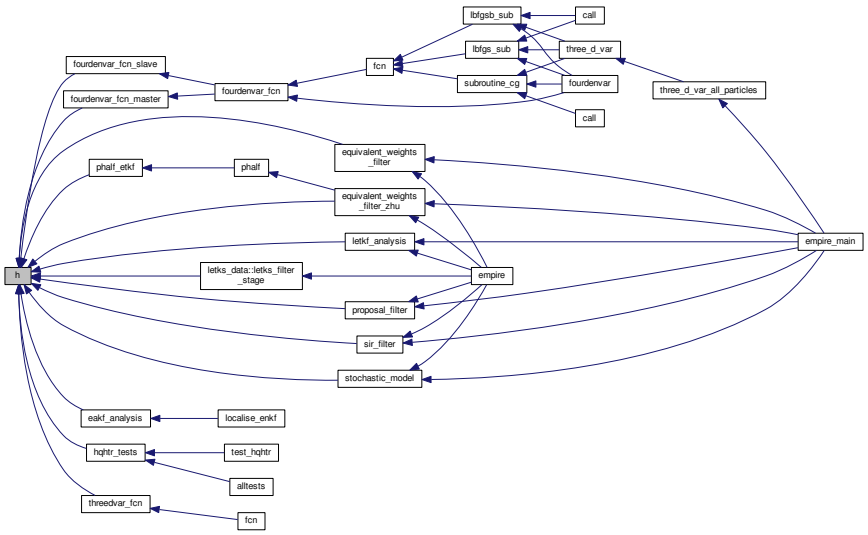
Given x compute Hx

Parameters

in	*obsdim*	the dimension of the observations
in	*nrhs*	the number of right hand sides
in	*x*	the input vectors in state space
out	*hx*	the resulting vector in observation space where hx $= Hx$
in	*t*	the timestep

Definition at line 221 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.6 subroutine ht (integer, intent(in) *obsDim,* integer, intent(in) *nrhs,* real(kind=rk), dimension(obsdim,nrhs), intent(in) *y,* real(kind=rk), dimension(state_dim,nrhs), intent(out) *x,* integer, intent(in) *t*)

subroutine to take an observation vector y and return x $= H^T(y)$ in full state space.

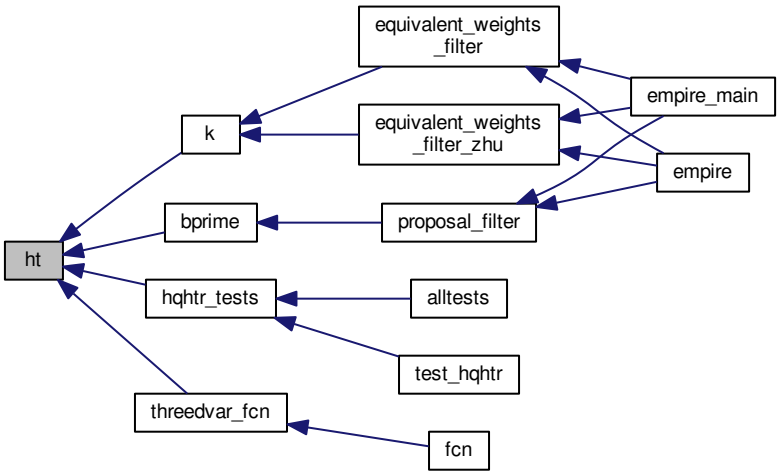
Given y compute $x = H^T(y)$

Parameters

in	*obsdim*	the dimension of the observations
in	*nrhs*	the number of right hand sides
in	*y*	the input vectors in observation space
out	*x*	the resulting vector in state space where x $= H^T y$
in	*t*	the timestep

Definition at line 243 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.7 subroutine q (integer, intent(in) *nrhs,* real(kind=rk), dimension(state_dim,nrhs), intent(in) *x,* real(kind=rk), dimension(state_dim,nrhs), intent(out) *Qx*)

subroutine to take a full state vector x and return Qx in state space.

Given x compute Qx

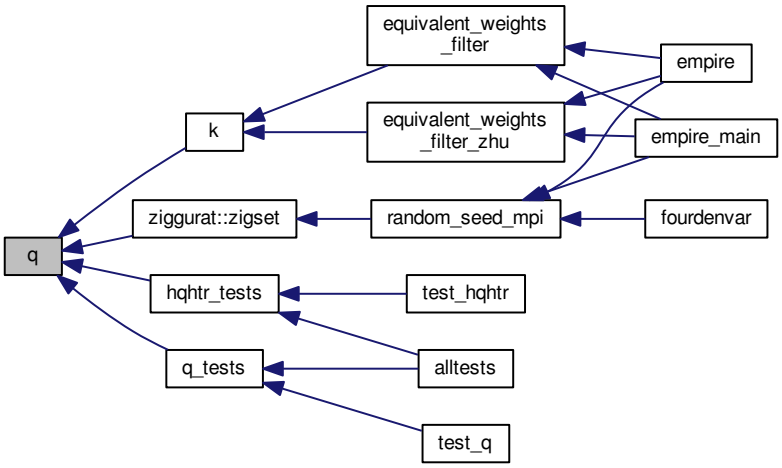
Parameters

in	*nrhs*	the number of right hand sides
in	*x*	the input vector
out	*qx*	the resulting vector where Qx $= Qx$

Definition at line 131 of file model_specific.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



12.8.1.8 subroutine qhalf (integer, intent(in) *nrhs,* real(kind=rk), dimension(state_dim,nrhs), intent(in) *x,* real(kind=rk), dimension(state_dim,nrhs), intent(out) *Qx*)

subroutine to take a full state vector x and return $Q^{1/2}x$ in state space.

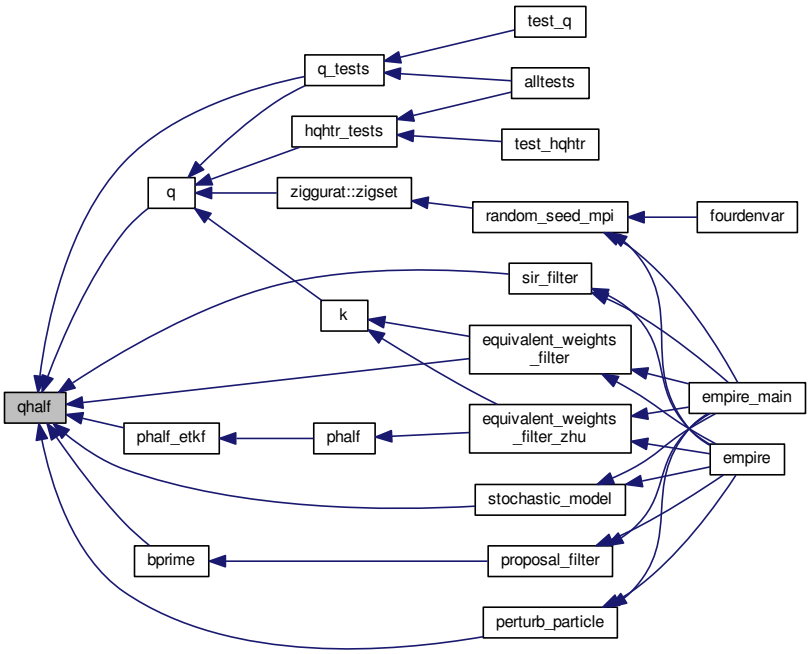
Given x compute $Q^{\frac{1}{2}}x$

Parameters

in	*nrhs*	the number of right hand sides
in	*x*	the input vector
out	*qx*	the resulting vector where Qx $= Q^{\frac{1}{2}}x$

Definition at line 156 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.9 subroutine r (integer, intent(in) *obsDim,* integer, intent(in) *nrhs,* real(kind=rk), dimension(obsdim,nrhs), intent(in) *y,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *Ry,* integer, intent(in) *t*)

subroutine to take an observation vector x and return Rx in observation space.

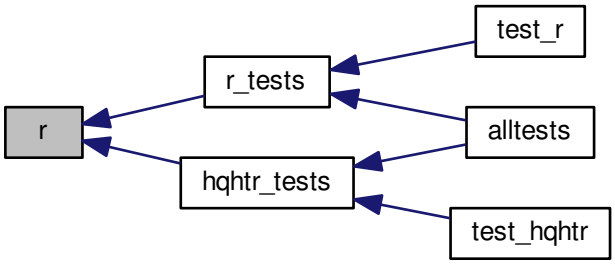
Given y compute Ry

Parameters

in	*obsdim*	the dimension of the observations
in	*nrhs*	the number of right hand sides
in	*y*	the input vector
out	*ry*	the resulting vectors where Ry $= Ry$
in	*t*	the timestep

Definition at line 176 of file model_specific.f90.

Here is the caller graph for this function:

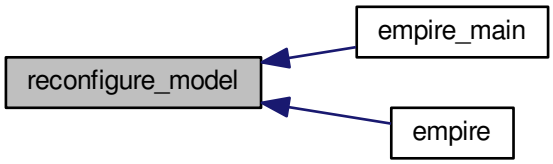


12.8.1.10 subroutine reconfigure_model ( )

subroutine to reset variables that may change when the observation network changes

Definition at line 58 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.11 subroutine rhalf (integer, intent(in) *obsDim,* integer, intent(in) *nrhs,* real(kind=rk), dimension(obsdim,nrhs), intent(in) *y,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *Ry,* integer, intent(in) *t*)

subroutine to take an observation vector x and return Rx in observation space.

Given y compute $R^{\frac{1}{2}}y$

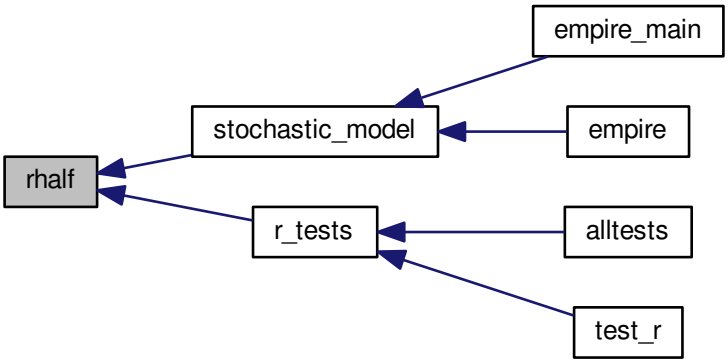
Parameters

in	*obsdim*	the dimension of the observations
in	*nrhs*	the number of right hand sides
in	*y*	the input vector

out	*ry*	the resulting vector where Ry $= R^{\frac{1}{2}}y$
in	*t*	the timestep

Definition at line 198 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.12 subroutine solve_b (integer, intent(in) *nrhs*, real(kind=rk), dimension(state_dim,nrhs), intent(in) *x*, real(kind=rk), dimension(state_dim,nrhs), intent(out) *v*)

subroutine to take a state vector x and return v in state space.

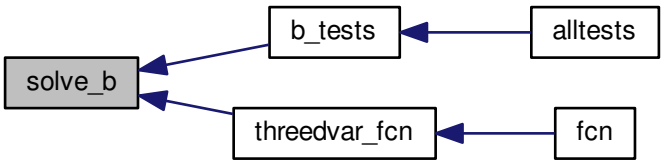
Given y find v such that $Bv = x$

Parameters

in	*nrhs*	the number of right hand sides
in	*x*	input vector
out	*v*	result vector where $v = B^{-1}x$

Definition at line 301 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.13 subroutine solve_hqht_plus_r (integer, intent(in) *obsdim,* real(kind=rk), dimension(obsdim), intent(in) *y,* real(kind=rk), dimension(obsdim), intent(out) *v,* integer, intent(in) *t*)

subroutine to take an observation vector y and return v in observation space.

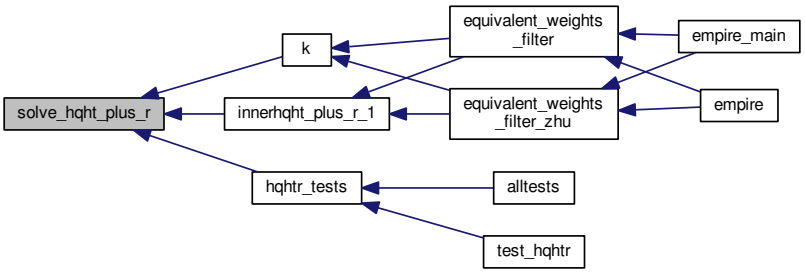
Given y find v such that $(HQH^T + R)v = y$

Parameters

in	*obsdim*	the dimension of the observations
in	*y*	the input vector
out	*v*	the result where $v = (HQH^T + R)^{-1}y$
in	*t*	the timestep

Definition at line 111 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.14 subroutine solve_r (integer, intent(in) *obsDim,* integer, intent(in) *nrhs,* real(kind=rk), dimension(obsdim,nrhs), intent(in) *y,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *v,* integer, intent(in) *t*)

subroutine to take an observation vector y and return v in observation space.

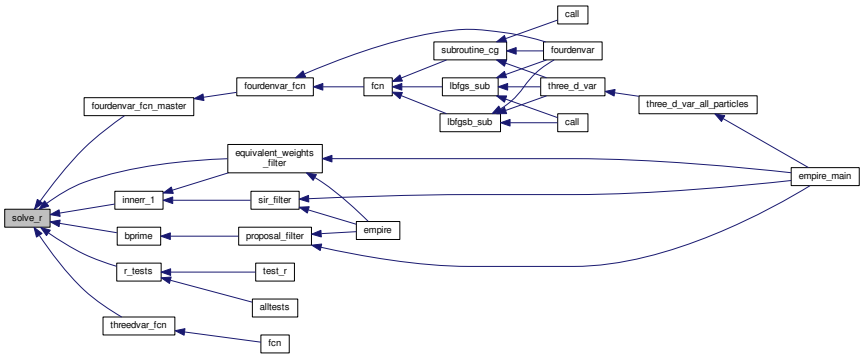
Given y find v such that $Rv = y$

Parameters

in	*obsdim*	the dimension of the observations
in	*nrhs*	the number of right hand sides
in	*y*	input vector
out	*v*	result vector where $v = R^{-1}y$
in	*t*	the timestep

Definition at line 68 of file model_specific.f90.

Here is the caller graph for this function:



12.8.1.15 subroutine solve_rhalf (integer, intent(in) *obsdim,* integer, intent(in) *nrhs,* real(kind=rk), dimension(obsdim,nrhs), intent(in) *y,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *v,* integer, intent(in) *t*)

subroutine to take an observation vector y and return v in observation space.

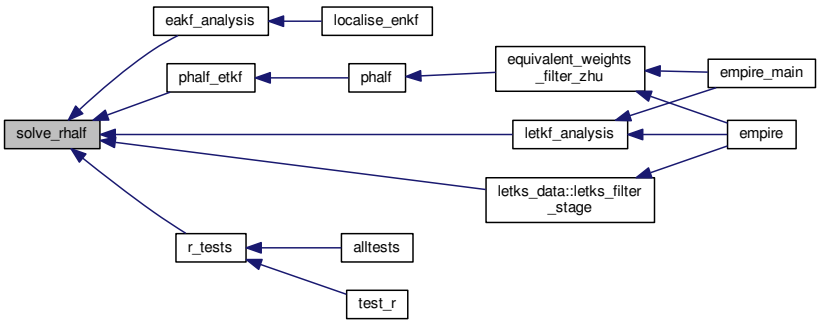
Given y find v such that $R^{\frac{1}{2}}v = y$

Parameters

in	*obsdim*	the dimension of the observations
in	*nrhs*	the number of right hand sides
in	*y*	input vector
out	*v*	result vector where $v = R^{-\frac{1}{2}}y$
in	*t*	the timestep

Definition at line 89 of file model_specific.f90.

Here is the caller graph for this function:



12.9 models/linear/linear_empire_vader.f90 File Reference

Functions/Subroutines

- program linear

program to implement a simple linear model of no use to anyone but for testing and debugging purposes :)

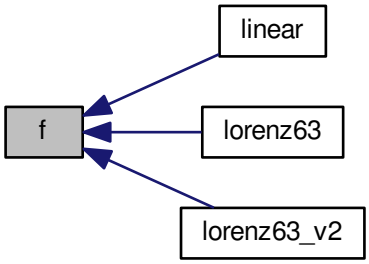
- real(kind=kind(1.0d0)) function, dimension(n) f (n, x)
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.9.1 Function/Subroutine Documentation

12.9.1.1 real(kind=kind(1.0d0)) function, dimension(n) linear::f (integer, intent(in) *n,* real(kind=kind(1.0d0)), dimension (n), intent(in) *x*)

Definition at line 112 of file linear_empire_vader.f90.

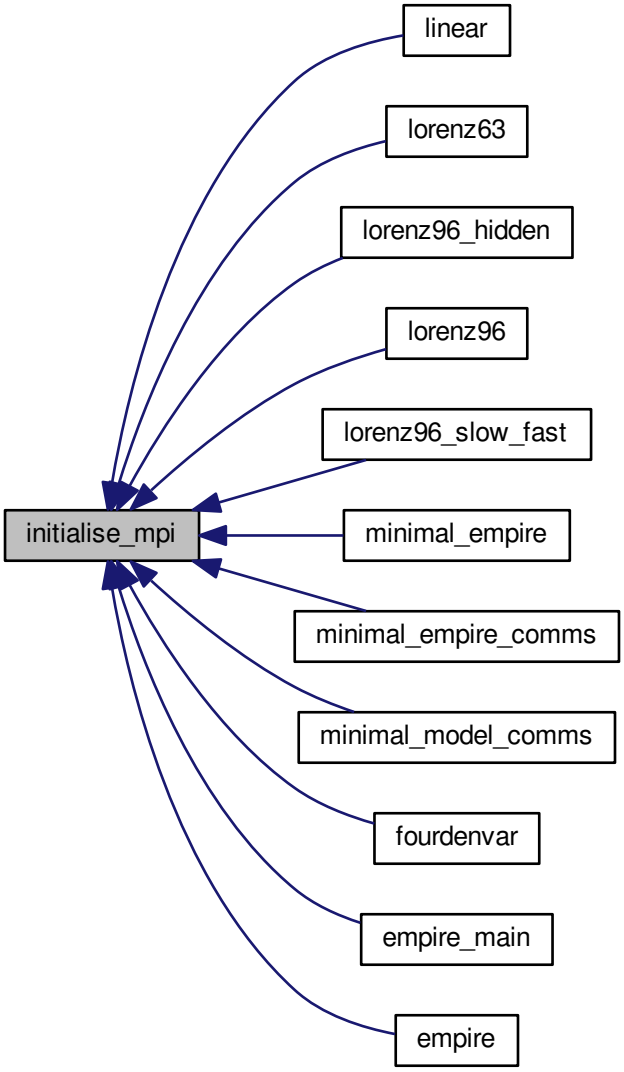
Here is the caller graph for this function:



12.9.1.2 subroutine linear::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 121 of file linear_empire_vader.f90.

Here is the caller graph for this function:



12.9.1.3 program linear ( )

program to implement a simple linear model of no use to anyone but for testing and debugging purposes :)

NOTE: THIS PROGRAM ***MUST*** RECIEVE A COUPLET OF INTEGERS FROM THE DATA ASSIMILATION CODE CONTAINING

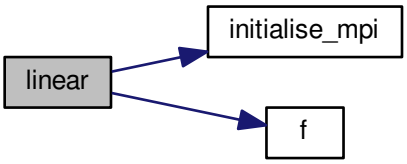
FIRST : THE SIZE OF THE DIMENSION OF THE MODEL

SECOND: THE NUMBER OF TIMESTEPS THE MODEL SHOULD DO

THIS IS A BIT WEIRD, AS NORMALLY THE MODEL DICTATES SUCH THINGS. BUT THIS IS A USELESS TOY MODEL. SO WE MIGHT AS WELL MAKE IT EASY TO USE TO TEST DA.

Definition at line 44 of file linear_empire_vader.f90.

Here is the call graph for this function:



12.10 models/linear/linear_empire_vader_v2.f90 File Reference

Functions/Subroutines

- program linear

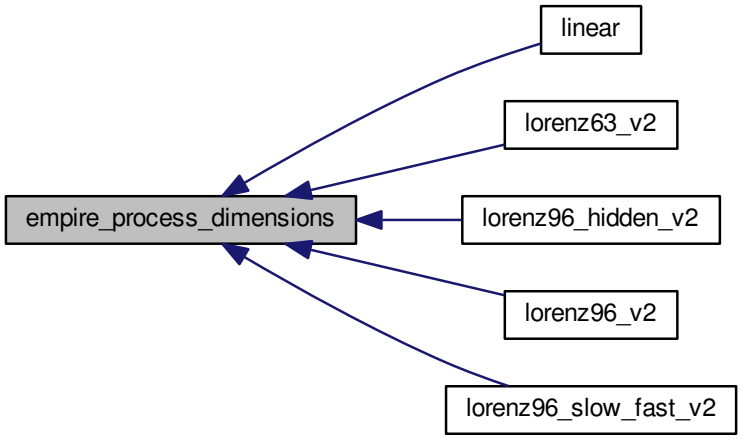
  program to implement a simple linear model of no use to anyone but for testing and debugging purposes :)
- real(kind=kind(1.0d0)) function, dimension(n) f (n, x)
- subroutine initialise_mpi_v2 (mdl_rank, cpl_root, cpl_mpi_comm)
- subroutine empire_process_dimensions (N, cpl_root, cpl_mpi_comm)

12.10.1 Function/Subroutine Documentation

12.10.1.1 subroutine linear::empire_process_dimensions (integer, intent(in) *N,* integer, intent(in) *cpl_root,* integer, intent(in) *cpl_mpi_comm*)

Definition at line 337 of file linear_empire_vader_v2.f90.

Here is the caller graph for this function:



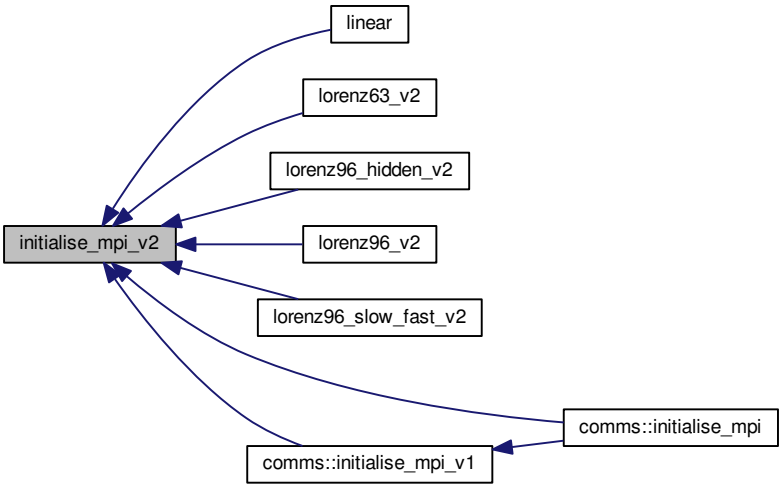
12.10.1.2 real(kind=kind(1.0d0)) function, dimension(n) linear::f (integer, intent(in) *n,* real(kind=kind(1.0d0)), dimension (n), intent(in) *x*)

Definition at line 128 of file linear_empire_vader_v2.f90.

12.10.1.3 subroutine linear::initialise_mpi_v2 (integer, intent(out) *mdl_rank,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 137 of file linear_empire_vader_v2.f90.

Here is the caller graph for this function:



12.10.1.4 program linear ( )

program to implement a simple linear model of no use to anyone but for testing and debugging purposes :)

NOTE: THIS PROGRAM ***MUST*** RECIEVE A COUPLET OF INTEGERS FROM THE DATA ASSIMILATION CODE CONTAINING

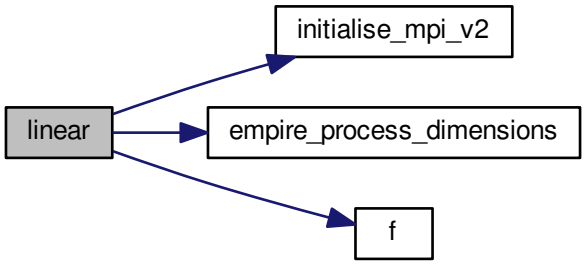
FIRST : THE SIZE OF THE DIMENSION OF THE MODEL

SECOND: THE NUMBER OF TIMESTEPS THE MODEL SHOULD DO

THIS IS A BIT WEIRD, AS NORMALLY THE MODEL DICTATES SUCH THINGS. BUT THIS IS A USELESS TOY MODEL. SO WE MIGHT AS WELL MAKE IT EASY TO USE TO TEST DA.

Definition at line 44 of file linear_empire_vader_v2.f90.

Here is the call graph for this function:



12.11 models/lorenz63/Lorenz63_empire.f90 File Reference

Functions/Subroutines

- program lorenz63
- real(kind=kind(1.0d0)) function, dimension(3) f (x, sigma, rho, beta)
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.11.1 Function/Subroutine Documentation

12.11.1.1 real(kind=kind(1.0d0)) function, dimension(3) lorenz63::f (real(kind=kind(1.0d0)), dimension (3), intent(in) *x,* real(kind=kind(1.0d0)), intent(in) *sigma,* real(kind=kind(1.0d0)), intent(in) *rho,* real(kind=kind(1.0d0)), intent(in) *beta*)

Definition at line 74 of file Lorenz63_empire.f90.

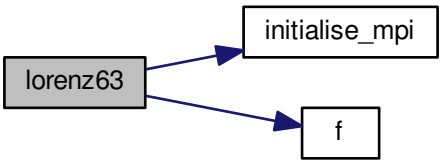
12.11.1.2 subroutine lorenz63::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 81 of file Lorenz63_empire.f90.

12.11.1.3 program lorenz63 ( )

Definition at line 29 of file Lorenz63_empire.f90.

Here is the call graph for this function:



12.12 models/lorenz63/Lorenz63_empire_v2.f90 File Reference

Functions/Subroutines

- program lorenz63_v2
- real(kind=kind(1.0d0)) function, dimension(3) f (x, sigma, rho, beta)
- subroutine initialise_mpi_v2 (mdl_rank, cpl_root, cpl_mpi_comm)
- subroutine empire_process_dimensions (N, cpl_root, cpl_mpi_comm)

12.12.1 Function/Subroutine Documentation

12.12.1.1 subroutine lorenz63_v2::empire_process_dimensions (integer, intent(in) *N,* integer, intent(in) *cpl_root,* integer, intent(in) *cpl_mpi_comm*)

Definition at line 298 of file Lorenz63_empire_v2.f90.

12.12.1.2 real(kind=kind(1.0d0)) function, dimension(3) lorenz63_v2::f (real(kind=kind(1.0d0)), dimension (3), intent(in) *x,* real(kind=kind(1.0d0)), intent(in) *sigma,* real(kind=kind(1.0d0)), intent(in) *rho,* real(kind=kind(1.0d0)), intent(in) *beta*)

Definition at line 90 of file Lorenz63_empire_v2.f90.

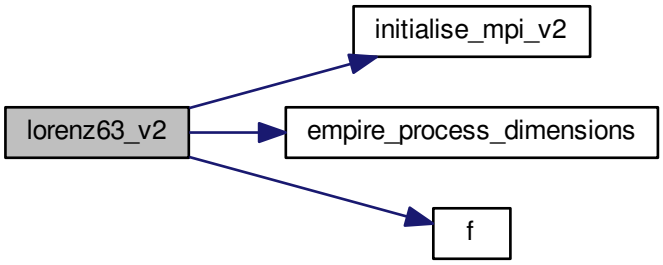
12.12.1.3 subroutine lorenz63_v2::initialise_mpi_v2 (integer, intent(out) *mdl_rank,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 98 of file Lorenz63_empire_v2.f90.

12.12.1.4 program lorenz63_v2 ( )

Definition at line 29 of file Lorenz63_empire_v2.f90.

Here is the call graph for this function:



12.13 models/lorenz96/hidden/Lorenz96_hidden_empire.f90 File Reference

Functions/Subroutines

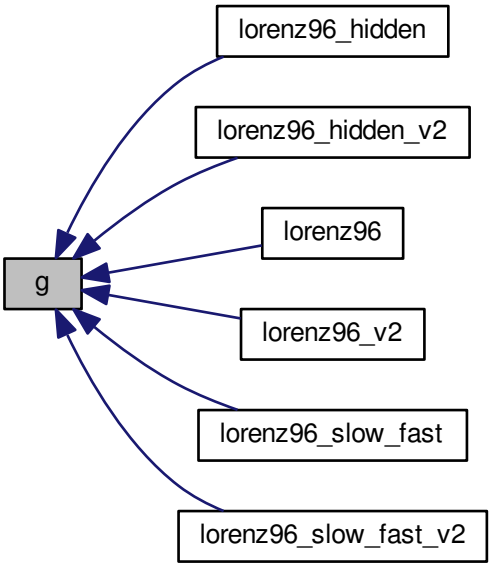
- program lorenz96_hidden
- real(kind=kind(1.0d0)) function, dimension(n, 3) g (X, N, F, alpha, delta, epsilon, gamma)
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.13.1 Function/Subroutine Documentation

12.13.1.1 real(kind=kind(1.0d0)) function, dimension(n,3) lorenz96_hidden::g (real(kind=kind(1.0d0)), dimension(n,3), intent(in) *X,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F,* real(kind=kind(1.0d0)), intent(in) *alpha,* real(kind=kind(1.0d0)), intent(in) *delta,* real(kind=kind(1.0d0)), intent(in) *epsilon,* real(kind=kind(1.0d0)), intent(in) *gamma*)

Definition at line 117 of file Lorenz96_hidden_empire.f90.

Here is the caller graph for this function:



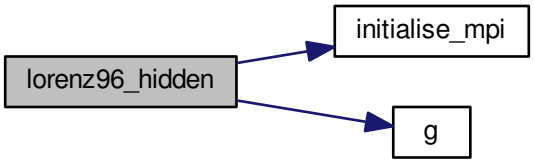
12.13.1.2 subroutine lorenz96_hidden::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 151 of file Lorenz96_hidden_empire.f90.

12.13.1.3 program lorenz96_hidden ( )

Definition at line 30 of file Lorenz96_hidden_empire.f90.

Here is the call graph for this function:



12.14 models/lorenz96/hidden/Lorenz96_hidden_empire_v2.f90 File Reference

Functions/Subroutines

- program lorenz96_hidden_v2
- real(kind=kind(1.0d0)) function, dimension(n, 3) g (X, N, F, alpha, delta, epsilon, gamma)
- subroutine initialise_mpi_v2 (mdl_rank, cpl_root, cpl_mpi_comm)
- subroutine empire_process_dimensions (N, cpl_root, cpl_mpi_comm)

12.14.1 Function/Subroutine Documentation

12.14.1.1 subroutine lorenz96_hidden_v2::empire_process_dimensions (integer, intent(in) *N,* integer, intent(in) *cpl_root,* integer, intent(in) *cpl_mpi_comm*)

Definition at line 365 of file Lorenz96_hidden_empire_v2.f90.

12.14.1.2 real(kind=kind(1.0d0)) function, dimension(n,3) lorenz96_hidden_v2::g (real(kind=kind(1.0d0)), dimension(n,3), intent(in) *X,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F,* real(kind=kind(1.0d0)), intent(in) *alpha,* real(kind=kind(1.0d0)), intent(in) *delta,* real(kind=kind(1.0d0)), intent(in) *epsilon,* real(kind=kind(1.0d0)), intent(in) *gamma*)

Definition at line 133 of file Lorenz96_hidden_empire_v2.f90.

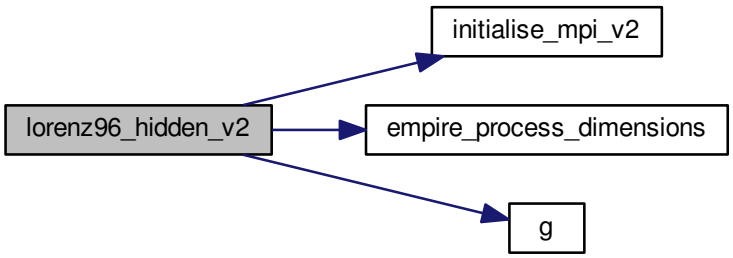
12.14.1.3 subroutine lorenz96_hidden_v2::initialise_mpi_v2 (integer, intent(out) *mdl_rank,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 167 of file Lorenz96_hidden_empire_v2.f90.

12.14.1.4 program lorenz96_hidden_v2 ( )

Definition at line 30 of file Lorenz96_hidden_empire_v2.f90.

Here is the call graph for this function:



12.15 models/lorenz96/Lorenz96_empire.f90 File Reference

Functions/Subroutines

- program lorenz96

- real(kind=kind(1.0d0)) function, dimension(0:n-1) g (x, N, F)
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.15.1 Function/Subroutine Documentation

12.15.1.1 real(kind=kind(1.0d0)) function, dimension(0:n-1) lorenz96::g (real(kind=kind(1.0d0)), dimension(0:n-1), intent(in) *x,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F*)

Definition at line 110 of file Lorenz96_empire.f90.

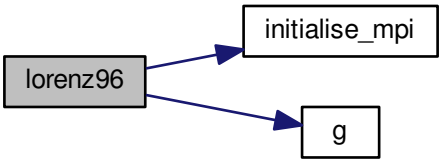
12.15.1.2 subroutine lorenz96::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 122 of file Lorenz96_empire.f90.

12.15.1.3 program lorenz96 ( )

Definition at line 29 of file Lorenz96_empire.f90.

Here is the call graph for this function:



12.16 models/lorenz96/Lorenz96_empire_v2.f90 File Reference

Functions/Subroutines

- program lorenz96_v2
- real(kind=kind(1.0d0)) function, dimension(0:n-1) g (x, N, F)
- subroutine initialise_mpi_v2 (mdl_rank, cpl_root, cpl_mpi_comm)
- subroutine empire_process_dimensions (N, cpl_root, cpl_mpi_comm)

12.16.1 Function/Subroutine Documentation

12.16.1.1 subroutine lorenz96_v2::empire_process_dimensions (integer, intent(in) *N,* integer, intent(in) *cpl_root,* integer, intent(in) *cpl_mpi_comm*)

Definition at line 328 of file Lorenz96_empire_v2.f90.

12.16.1.2 real(kind=kind(1.0d0)) function, dimension(0:n-1) lorenz96_v2::g (real(kind=kind(1.0d0)), dimension(0:n-1), intent(in) *x,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F*)

Definition at line 118 of file Lorenz96_empire_v2.f90.

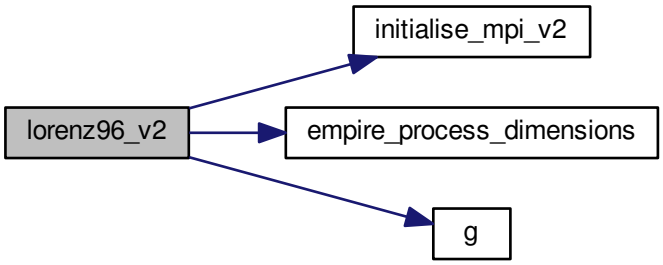
12.16.1.3 subroutine lorenz96_v2::initialise_mpi_v2 (integer, intent(out) *mdl_rank,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 130 of file Lorenz96_empire_v2.f90.

12.16.1.4 program lorenz96_v2 ( )

Definition at line 29 of file Lorenz96_empire_v2.f90.

Here is the call graph for this function:



12.17 models/lorenz96/slow_fast/Lorenz96_slow_fast.f90 File Reference

Functions/Subroutines

- program lorenz96_slow_fast
- real(kind=kind(1.0d0)) function, dimension(n, 3) g (X, N, F, alpha, delta, epsilon, gamma)
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.17.1 Function/Subroutine Documentation

12.17.1.1 real(kind=kind(1.0d0)) function, dimension(n,3) lorenz96_slow_fast::g (real(kind=kind(1.0d0)), dimension(n,3), intent(in) *X,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F,* real(kind=kind(1.0d0)), intent(in) *alpha,* real(kind=kind(1.0d0)), intent(in) *delta,* real(kind=kind(1.0d0)), intent(in) *epsilon,* real(kind=kind(1.0d0)), intent(in) *gamma*)

Definition at line 117 of file Lorenz96_slow_fast.f90.

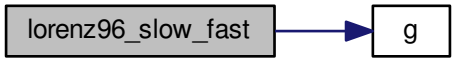
12.17.1.2 subroutine lorenz96_slow_fast::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 151 of file Lorenz96_slow_fast.f90.

12.17.1.3 program lorenz96_slow_fast ( )

Definition at line 30 of file Lorenz96_slow_fast.f90.

Here is the call graph for this function:



12.18 models/lorenz96/slow_fast/Lorenz96_slow_fast_empire.f90 File Reference

Functions/Subroutines

- program lorenz96_slow_fast
- real(kind=kind(1.0d0)) function, dimension(n, 3) g (X, N, F, alpha, delta, epsilon, gamma)
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.18.1 Function/Subroutine Documentation

12.18.1.1 real(kind=kind(1.0d0)) function, dimension(n,3) lorenz96_slow_fast::g (real(kind=kind(1.0d0)), dimension(n,3), intent(in) *X,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F,* real(kind=kind(1.0d0)), intent(in) *alpha,* real(kind=kind(1.0d0)), intent(in) *delta,* real(kind=kind(1.0d0)), intent(in) *epsilon,* real(kind=kind(1.0d0)), intent(in) *gamma*)

Definition at line 117 of file Lorenz96_slow_fast_empire.f90.

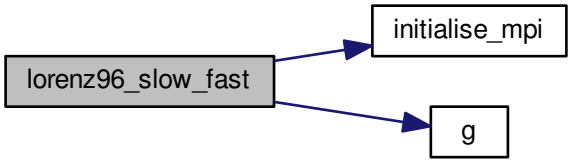
12.18.1.2 subroutine lorenz96_slow_fast::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 151 of file Lorenz96_slow_fast_empire.f90.

12.18.1.3 program lorenz96_slow_fast ( )

Definition at line 30 of file Lorenz96_slow_fast_empire.f90.

Here is the call graph for this function:



12.19 models/lorenz96/slow_fast/Lorenz96_slow_fast_empire_v2.f90 File Reference

Functions/Subroutines

- program lorenz96_slow_fast_v2
- real(kind=kind(1.0d0)) function, dimension(n, 3) g (X, N, F, alpha, delta, epsilon, gamma)
- subroutine initialise_mpi_v2 (mdl_rank, cpl_root, cpl_mpi_comm)
- subroutine empire_process_dimensions (N, cpl_root, cpl_mpi_comm)

12.19.1 Function/Subroutine Documentation

12.19.1.1 subroutine lorenz96_slow_fast_v2::empire_process_dimensions (integer, intent(in) *N,* integer, intent(in) *cpl_root,* integer, intent(in) *cpl_mpi_comm*)

Definition at line 365 of file Lorenz96_slow_fast_empire_v2.f90.

12.19.1.2 real(kind=kind(1.0d0)) function, dimension(n,3) lorenz96_slow_fast_v2::g (real(kind=kind(1.0d0)), dimension(n,3), intent(in) *X,* integer, intent(in) *N,* real(kind=kind(1.0d0)), intent(in) *F,* real(kind=kind(1.0d0)), intent(in) *alpha,* real(kind=kind(1.0d0)), intent(in) *delta,* real(kind=kind(1.0d0)), intent(in) *epsilon,* real(kind=kind(1.0d0)), intent(in) *gamma*)

Definition at line 133 of file Lorenz96_slow_fast_empire_v2.f90.

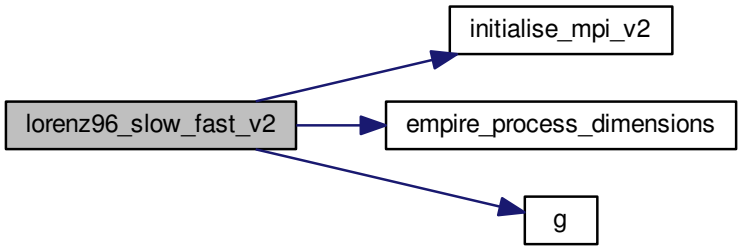
12.19.1.3 subroutine lorenz96_slow_fast_v2::initialise_mpi_v2 (integer, intent(out) *mdl_rank,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 167 of file Lorenz96_slow_fast_empire_v2.f90.

12.19.1.4 program lorenz96_slow_fast_v2 ( )

Definition at line 30 of file Lorenz96_slow_fast_empire_v2.f90.

Here is the call graph for this function:



12.20 models/minimal_empire/minimal_empire.f90 File Reference

Functions/Subroutines

- program minimal_empire
  the main program

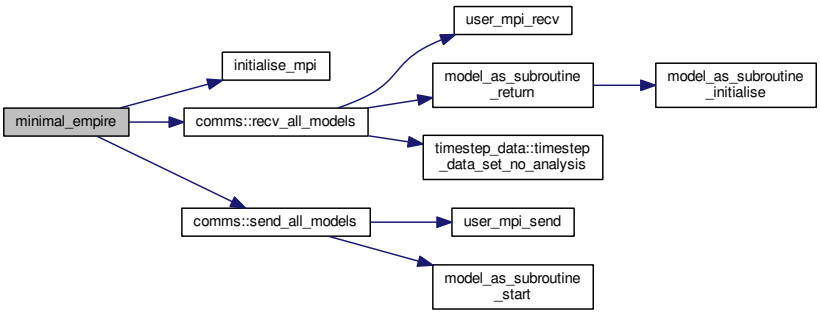
12.20.1 Function/Subroutine Documentation

12.20.1.1 program minimal_empire ( )

the main program

Definition at line 30 of file minimal_empire.f90.

Here is the call graph for this function:



12.21 models/minimal_empire_comms/minimal_empire_comms.f90 File Reference

Functions/Subroutines

- program minimal_empire_comms

the main program

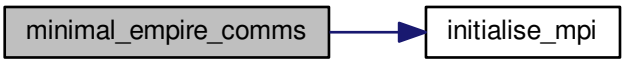
12.21.1 Function/Subroutine Documentation

12.21.1.1 program minimal_empire_comms ( )

the main program

Definition at line 30 of file minimal_empire_comms.f90.

Here is the call graph for this function:



12.22 models/minimal_model/minimal_model.f90 File Reference

Functions/Subroutines

- program minimal_model_comms
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.22.1 Function/Subroutine Documentation

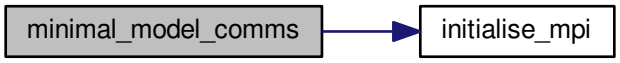
12.22.1.1 subroutine minimal_model_comms::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 78 of file minimal_model.f90.

12.22.1.2 program minimal_model_comms ( )

Definition at line 30 of file minimal_model.f90.

Here is the call graph for this function:



12.23 models/minimal_model/minimal_model_v2.f90 File Reference

Functions/Subroutines

- program minimal_model_comms_v2

12.23.1 Function/Subroutine Documentation

12.23.1.1 program minimal_model_comms_v2 ( )

Definition at line 30 of file minimal_model_v2.f90.

12.24 models/minimal_model/minimal_model_v3.f90 File Reference

Functions/Subroutines

- program minimal_model_v3

12.24.1 Function/Subroutine Documentation

12.24.1.1 program minimal_model_v3 ( )

Definition at line 30 of file minimal_model_v3.f90.

12.25 models/minimal_model_comms/minimal_model_comms.f90 File Reference

Functions/Subroutines

- program minimal_model_comms
- subroutine initialise_mpi (mdl_id, cpl_root, cpl_mpi_comm)

12.25.1 Function/Subroutine Documentation

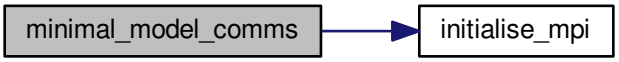
12.25.1.1 subroutine minimal_model_comms::initialise_mpi (integer, intent(out) *mdl_id,* integer, intent(out) *cpl_root,* integer, intent(out) *cpl_mpi_comm*)

Definition at line 37 of file minimal_model_comms.f90.

12.25.1.2 program minimal_model_comms ( )

Definition at line 30 of file minimal_model_comms.f90.

Here is the call graph for this function:



12.26 models/minimal_model_comms/minimal_model_comms_v2.f90 File Reference

Functions/Subroutines

- program minimal_model_comms_v2

12.26.1 Function/Subroutine Documentation

12.26.1.1 program minimal_model_comms_v2 ( )

Definition at line 30 of file minimal_model_comms_v2.f90.

12.27 models/minimal_model_comms/minimal_model_comms_v3.f90 File Reference

Functions/Subroutines

- program minimal_model_comms_v3

12.27.1 Function/Subroutine Documentation

12.27.1.1 program minimal_model_comms_v3 ( )

Definition at line 30 of file minimal_model_comms_v3.f90.

12.28 models/minimal_model_comms/minimal_model_comms_v5.f90 File Reference

Functions/Subroutines

- program minimal_model_comms_v5

12.28.1 Function/Subroutine Documentation

12.28.1.1 program minimal_model_comms_v5 ( )

Definition at line 30 of file minimal_model_comms_v5.f90.

12.29 src/4dEnVar/4dEnVar.f90 File Reference

Functions/Subroutines

- program fourdenvar

  the main program to run 4DEnVar

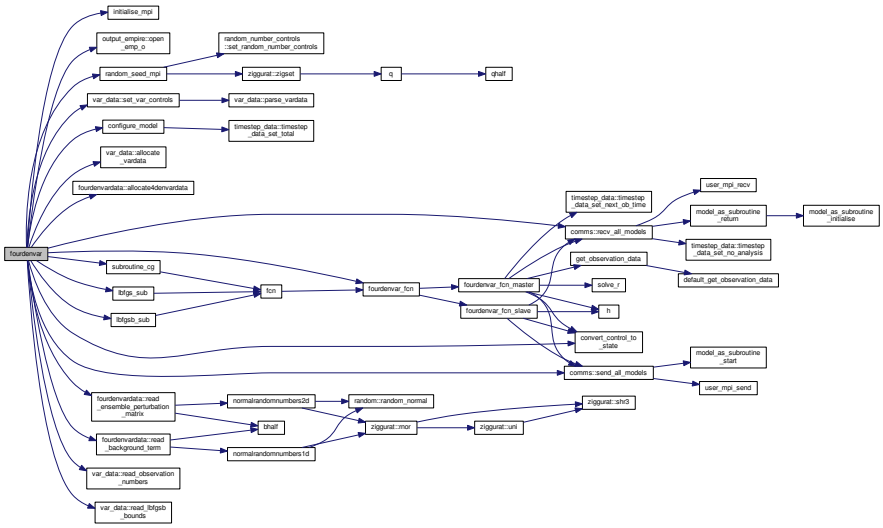
12.29.1 Function/Subroutine Documentation

12.29.1.1 program fourdenvar ( )

the main program to run 4DEnVar

Definition at line 29 of file 4dEnVar.f90.

Here is the call graph for this function:



12.30 src/4dEnVar/4denvar_fcn.f90 File Reference

Functions/Subroutines

- subroutine fcn (n, x, f, g)

  This is the subroutine which the optimization routines call to get the objective function value and its gradient.

- subroutine fourdenvar_fcn (n, v, f, g)

  subroutine to provide the objective function and gradient for 4dEnVar.

- subroutine convert_control_to_state (n, v, stateDim, x)

  a subroutine to convert the optimization control variable to a model state vector

- subroutine fourdenvar_fcn_master (n, v, f, g, leave)
- subroutine fourdenvar_fcn_slave (n, v, leave)

12.30.1 Function/Subroutine Documentation

12.30.1.1 subroutine convert_control_to_state (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(in) *v,* integer, intent(in) *stateDim,* real(kind=rk), dimension(statedim), intent(out) *x*)

a subroutine to convert the optimization control variable to a model state vector

this must be called by all processes on pf_mpi_comm

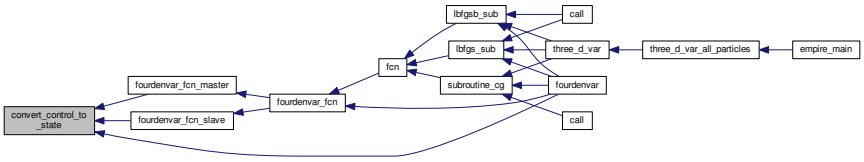
and the result x is known to all processes

Parameters

in	*n*	the dimension of the control variable
in	*v*	the optimization control variable
in	*statedim*	the dimension of the model state
out	*x*	the resulting model state

Definition at line 153 of file 4denvar_fcn.f90.

Here is the caller graph for this function:



12.30.1.2 subroutine fcn (integer, intent(in) *n,* real(kind=kind(1.0d0)), dimension(n), intent(in) *x,* real(kind=kind(1.0d0)), intent(out) *f,* real(kind=kind(1.0d0)), dimension(n), intent(out) *g*)

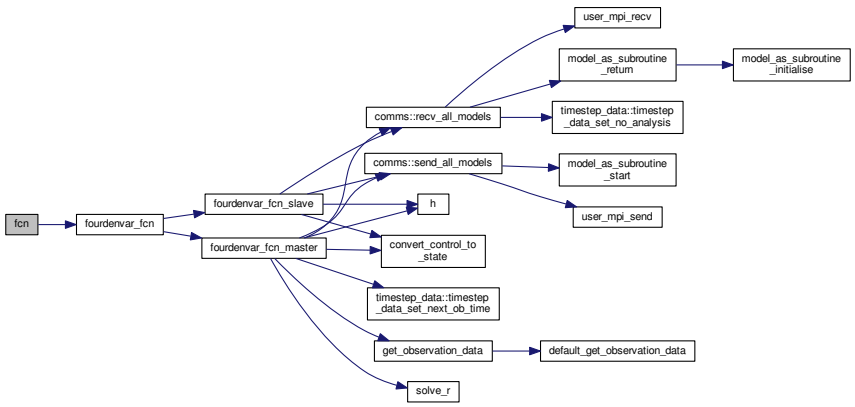
This is the subroutine which the optimization routines call to get the objective function value and its gradient.

Parameters

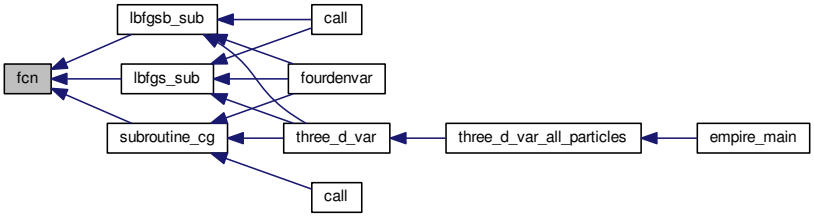
in	*n*	the dimension of the optimzation problem
in	*x*	the current optimization state
out	*f*	the objective function value
out	*g*	the gradient of the objective function

Definition at line 31 of file 4denvar_fcn.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.30.1.3 subroutine fourdenvar_fcn (integer, intent(in) *n*, real(kind=rk), dimension(n), intent(in) *v*, real(kind=rk), intent(out) *f*, real(kind=kind(1.0d0)), dimension(n), intent(out) *g*)

subroutine to provide the objective function and gradient for 4dEnVar.

Let x be the state we wish to find using Var.

Then we define $X_k := \{x_1(k) - x(k); \ldots; x_m(k) - x(k)\}$

to be the ensemble perturbation matrix,

where $x_j(k)$ is the jth ensemble member at time k

and $x(k)$ is the optimization solution integrated forward in time to timestep k.

The objective function considered is

$J(x) = \frac{1}{2}(x - x_b)^T B^{-1}(x - x_b) + \frac{1}{2}\sum_i (y_i - H_i(M_i(x)))^T R_i^{-1}(y_i - H_i(M_i(x)))$

where x_b is a background guess, B the background error covariance matrix,

y_i observations at a timestep i, H_i the corresponding observation operator with associated observation error covariance matrix R_i and M_i the model which propogates a state from time 0 to the observation timestep i.

In this code, $B := \frac{1}{m-1} X_0 X_0^T$.

We make the following control variable transform:

$x = X_0 v + x_b$ where $v \in \mathbb{R}^m$.

Then the objective function can be re-written as a function of v,

$f = J(x) = J(v) = \frac{1}{2}(m-1)v^T v + \frac{1}{2}\sum_i (y_i - H_i(M_i(X_0 v + x_b)))^T R_i^{-1}(y_i - H_i(M_i(X_0 v + x_b)))$.

The gradient of the objective function can then be written

$g = \nabla_v J(v) \approx (m-1)v - \sum_i (H_i(X_i))^T R_i^{-1}(y_i - H_i(M_i(X_0 v + x_b))$

which is exact if H_i and M_i are linear and X_0 is invertible (at least I assume there has to be this condition on X_0...).

Note this is not exactly the 4dEnVar algorithm as given by Liu, Xian and Wang (2008) as the ensemble perturbation matrix here are perturbations around the current var solution, not the perturbations around $M_i(x_b)$. i.e. $X_i = X_i(x) = X_i(v)$.

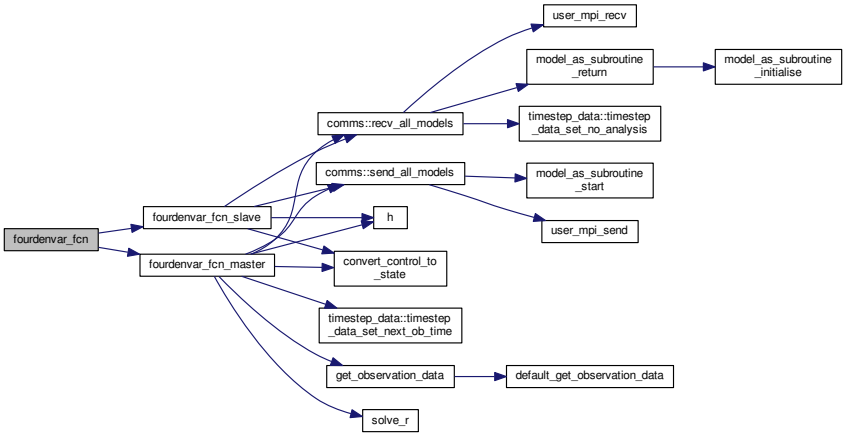
Parameters

in	*n*	this is the dimension of the optimization control variable, so the number of ensemble members-1

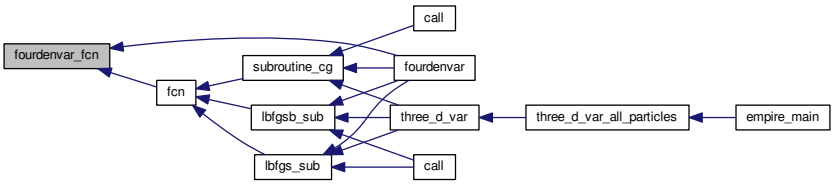
in	*v*	this is the optimization control variable
out	*f*	the 4dvar objective function
out	*g*	the gradient of the 4dvar objective function

Definition at line 105 of file 4denvar_fcn.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



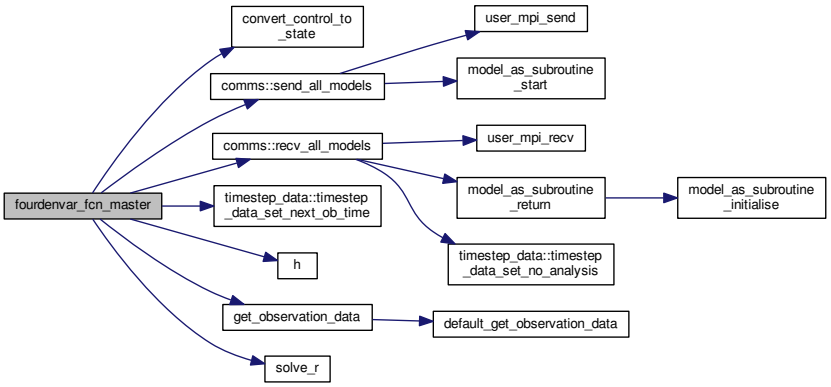
12.30.1.4 subroutine fourdenvar_fcn_master (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(in) *v,* real(kind=rk), intent(out) *f,* real(kind=kind(1.0d0)), dimension(n), intent(out) *g,* logical, intent(out) *leave*)

Parameters

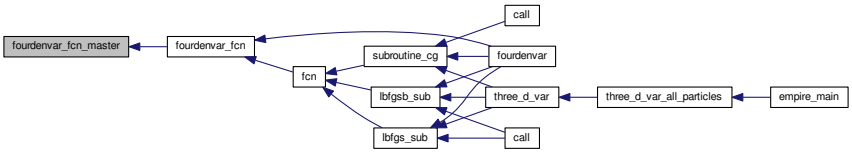
in	*n*	this is the dimension of the optimization control variable, so the number of ensemble members-1
in	*v*	this is the optimization control variable
out	*f*	the 4dvar objective function
out	*g*	the gradient of the 4dvar objective function

Definition at line 198 of file 4denvar_fcn.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



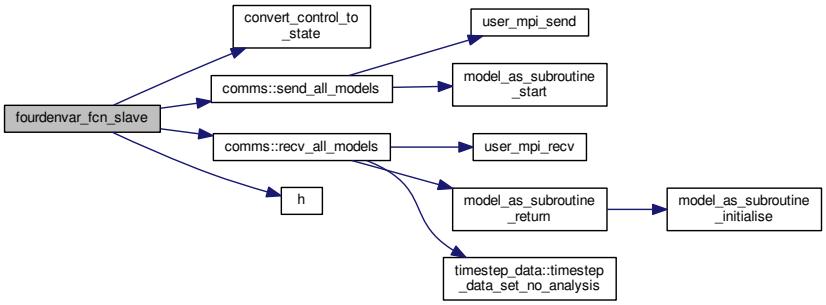
12.30.1.5 subroutine fourdenvar_fcn_slave (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(in) *v,* logical, intent(out) *leave*)

Parameters

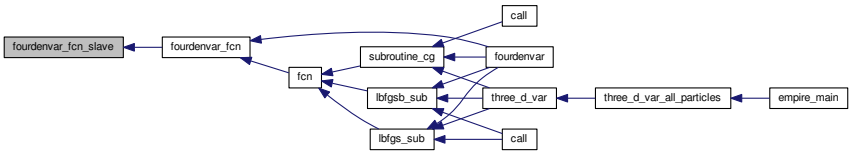
in	*n*	this is the dimension of the optimization control variable, so the number of ensemble members-1
in	*v*	this is the optimization control variable

Definition at line 402 of file 4denvar_fcn.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.31 src/4dEnVar/fourdenvardata.f90 File Reference

Data Types

- module fourdenvardata

  module holding data specific for 4denvar, not var itself. this is necessary because of the difference in x in optimization and in the model state.

12.32 src/4dEnVar/var_data.f90 File Reference

Data Types

- module var_data

  module holding data for variational problems
- type var_data::var_control_type

12.33 src/controllers/compile_options.f90 File Reference

Data Types

- module compile_options

  Module that stores logical variables to control the compilation.

12.34 src/controllers/empire.nml File Reference

12.35 src/controllers/empire_main.f90 File Reference

Functions/Subroutines

- program empire_main

  the main program

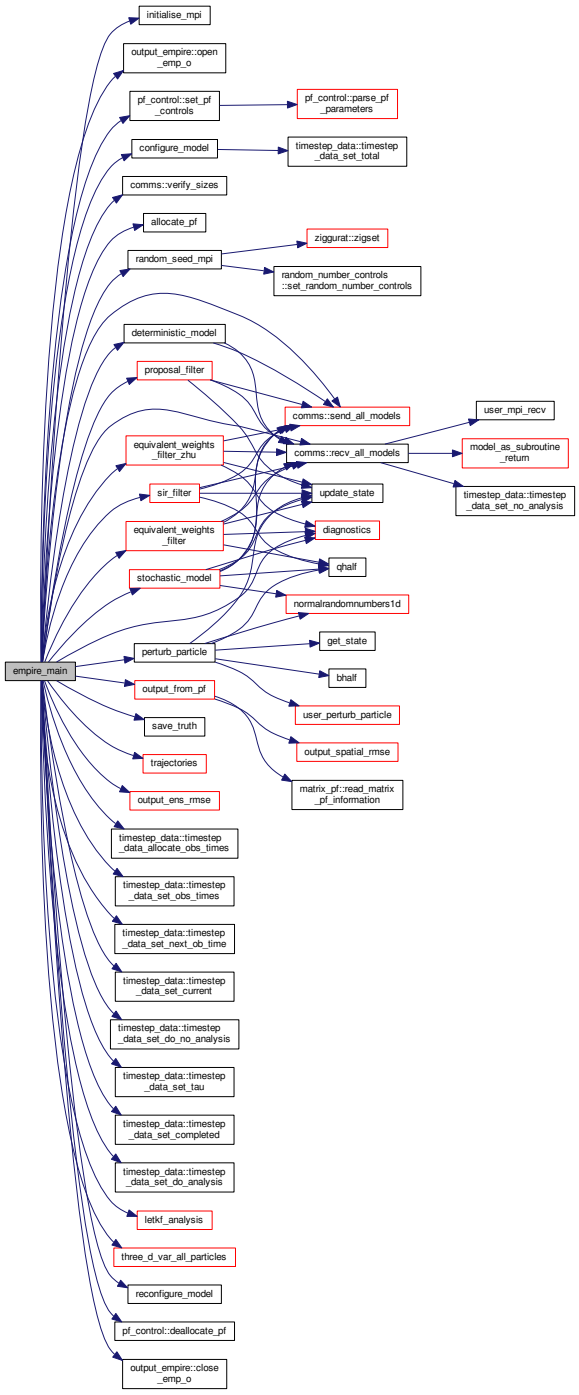
12.35.1 Function/Subroutine Documentation

12.35.1.1 program empire_main ( )

the main program

Definition at line 36 of file empire_main.f90.

Here is the call graph for this function:



12.36 src/controllers/letks_test.f90 File Reference

Functions/Subroutines

- program empire

the main program

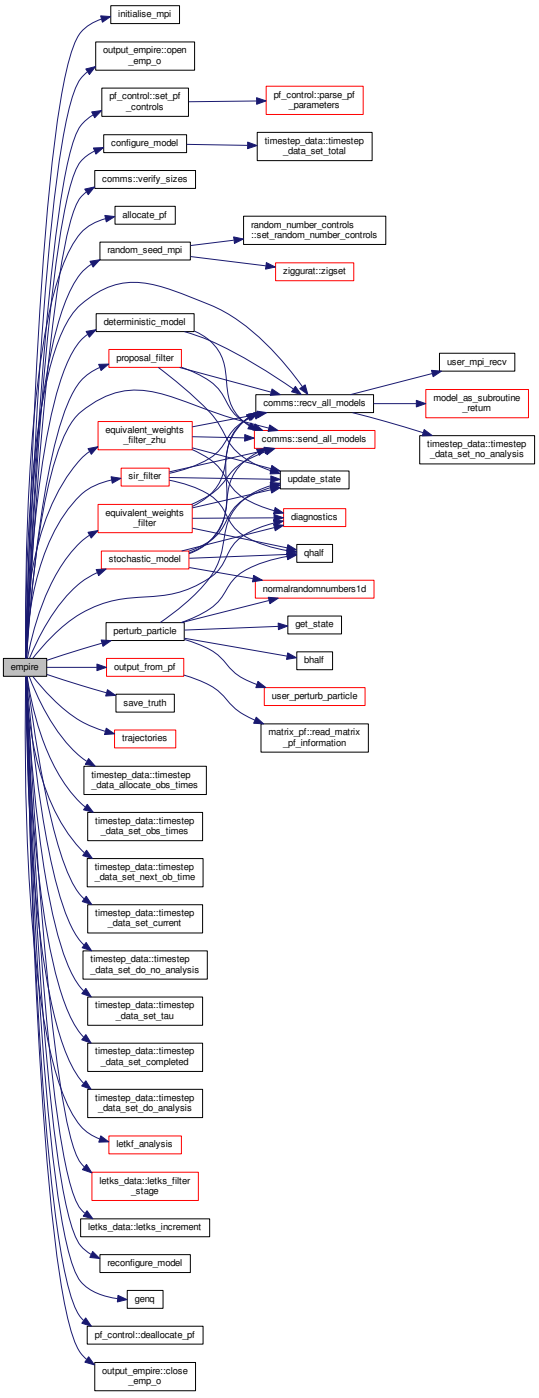
12.36.1 Function/Subroutine Documentation

12.36.1.1 program empire ( )

the main program

Definition at line 36 of file letks_test.f90.

Here is the call graph for this function:



12.37 src/controllers/output_empire.f90 File Reference

Data Types

- module output_empire

  Module that stores the information about the outputting from empire.

12.38 src/controllers/pf_control.f90 File Reference

Data Types

- module pf_control

  module pf_control holds all the information to control the the main program
- type pf_control::pf_control_type

12.39 src/controllers/sizes.f90 File Reference

Data Types

- module sizes

  Module that stores the dimension of observation and state spaces.

12.40 src/controllers/timestep_data.f90 File Reference

Data Types

- module timestep_data

  Module that stores the information about the timestepping process.
- type timestep_data::timestep_data_type

12.41 src/DOC_README.txt File Reference

12.42 src/DOC_VERSIONS.txt File Reference

12.43 src/filters/deterministic_model.f90 File Reference

Functions/Subroutines

- subroutine deterministic_model

  subroutine to simply move the model forward in time one timestep

12.43.1 Function/Subroutine Documentation

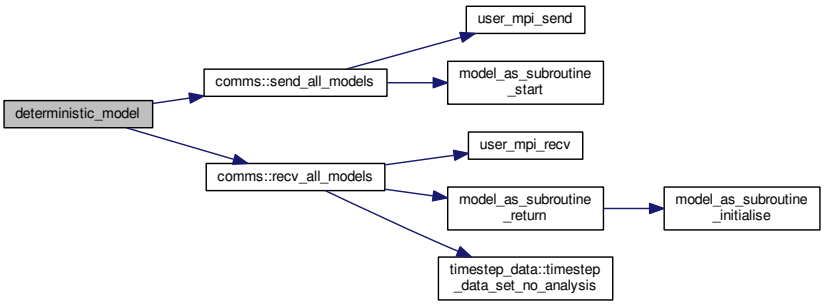
12.43.1.1 subroutine deterministic_model ( )

subroutine to simply move the model forward in time one timestep

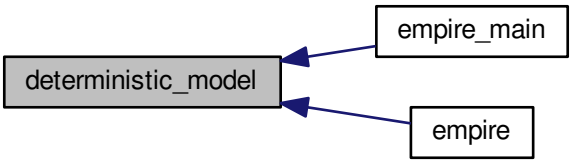
PAB 21-05-2013

Definition at line 33 of file deterministic_model.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.44 src/filters/eakf_analysis.f90 File Reference

Functions/Subroutines

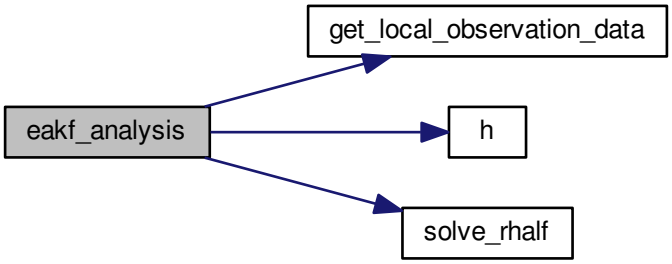
- subroutine eakf_analysis (num_hor, num_ver, this_hor, this_ver, boundary, x, N, stateDim, obsDim, rho)

12.44.1 Function/Subroutine Documentation

12.44.1.1 subroutine eakf_analysis (integer, intent(in) *num_hor,* integer, intent(in) *num_ver,* integer, intent(in) *this_hor,* integer, intent(in) *this_ver,* integer, intent(in) *boundary,* real(kind=rk), dimension(statedim,n), intent(inout) *x,* integer, intent(in) *N,* integer, intent(in) *stateDim,* integer, intent(in) *obsDim,* real(kind=rk), intent(in) *rho*)

Definition at line 27 of file eakf_analysis.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

12.45 src/filters/enkf_specific.f90 File Reference

Functions/Subroutines

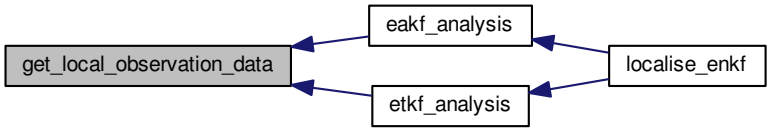
- subroutine h_local (num_hor, num_ver, this_hor, this_ver, boundary, nrhs, stateDim, x, obsDim, y)
- subroutine solve_rhalf_local (num_hor, num_ver, this_hor, this_ver, boundary, nrhs, obsDim, y, v)
- subroutine get_local_observation_data (num_hor, num_ver, this_hor, this_ver, boundary, obsDim, y)
- subroutine localise_enkf (enkf_analysis)

12.45.1 Function/Subroutine Documentation

12.45.1.1 subroutine get_local_observation_data (integer, intent(in) *num_hor,* integer, intent(in) *num_ver,* integer, intent(in) *this_hor,* integer, intent(in) *this_ver,* integer, intent(in) *boundary,* integer, intent(in) *obsDim,* real(kind=rk), dimension(obsdim), intent(out) *y*)

Definition at line 83 of file enkf_specific.f90.

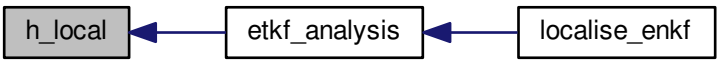
Here is the caller graph for this function:



12.45.1.2 subroutine h_local (integer, intent(in) *num_hor,* integer, intent(in) *num_ver,* integer, intent(in) *this_hor,* integer, intent(in) *this_ver,* integer, intent(in) *boundary,* integer, intent(in) *nrhs,* integer, intent(in) *stateDim,* real(kind=rk), dimension(statedim,nrhs), intent(in) *x,* integer, intent(in) *obsDim,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *y*)

Definition at line 27 of file enkf_specific.f90.

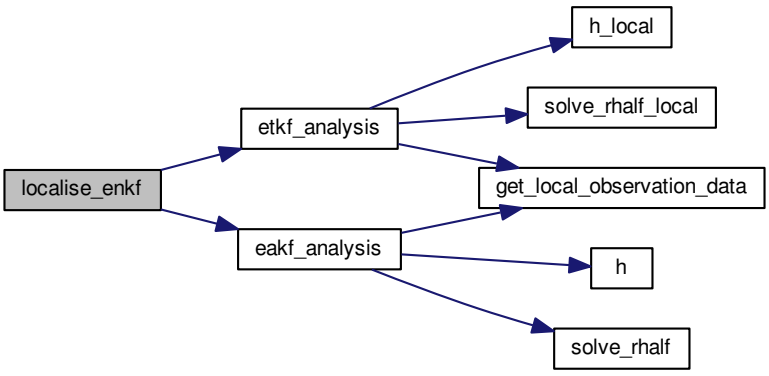
Here is the caller graph for this function:



12.45.1.3 subroutine localise_enkf (integer, intent(in) *enkf_analysis*)

Definition at line 142 of file enkf_specific.f90.

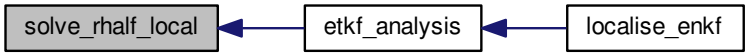
Here is the call graph for this function:



12.45.1.4 subroutine solve_rhalf_local (integer, intent(in) *num_hor,* integer, intent(in) *num_ver,* integer, intent(in) *this_hor,* integer, intent(in) *this_ver,* integer, intent(in) *boundary,* integer, intent(in) *nrhs,* integer, intent(in) *obsDim,* real(kind=rk), dimension(obsdim,nrhs), intent(in) *y,* real(kind=rk), dimension(obsdim,nrhs), intent(out) *v*)

Definition at line 69 of file enkf_specific.f90.

Here is the caller graph for this function:



12.46 src/filters/equivalent_weights_filter.f90 File Reference

Functions/Subroutines

- subroutine equivalent_weights_filter

  subroutine to do the equivalent weights step

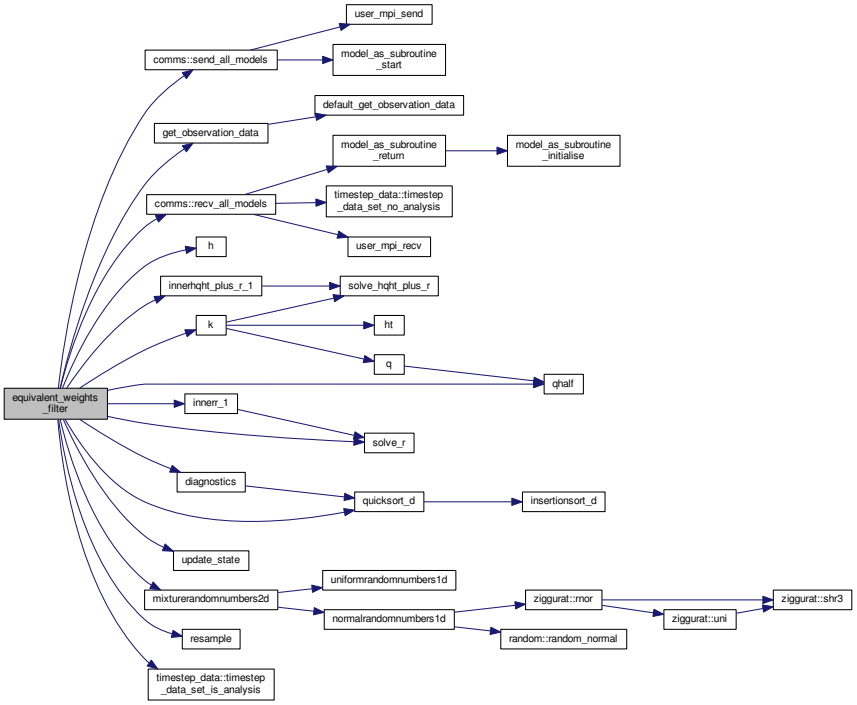
12.46.1 Function/Subroutine Documentation

12.46.1.1 subroutine equivalent_weights_filter ( )

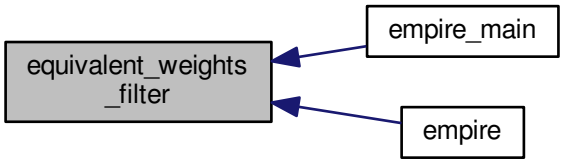
subroutine to do the equivalent weights step

Definition at line 29 of file equivalent_weights_filter.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.47 src/filters/equivalent_weights_filter_zhu.f90 File Reference

Functions/Subroutines

- subroutine equivalent_weights_filter_zhu

  subroutine to do the new scheme equal weights last time-step

12.47.1 Function/Subroutine Documentation

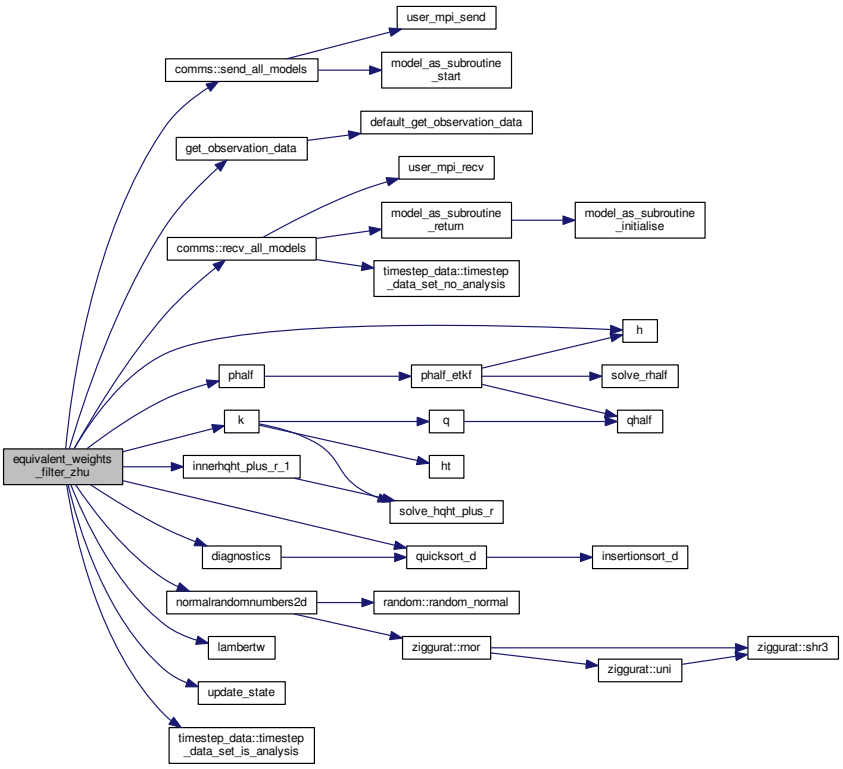
12.47.1.1 subroutine equivalent_weights_filter_zhu ( )

subroutine to do the new scheme equal weights last time-step

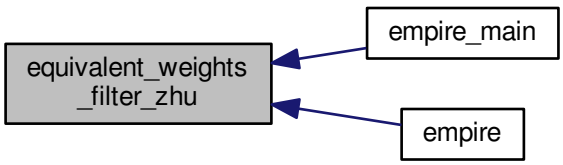
structure of code loosely based on original equivalent weights scheme equivalent_weights_filter

Definition at line 32 of file equivalent_weights_filter_zhu.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.48 src/filters/etkf_analysis.f90 File Reference

Functions/Subroutines

- subroutine etkf_analysis (num_hor, num_ver, this_hor, this_ver, boundary, x, N, stateDim, obsDim, rho)

  subroutine to perform the ensemble transform Kalman filter

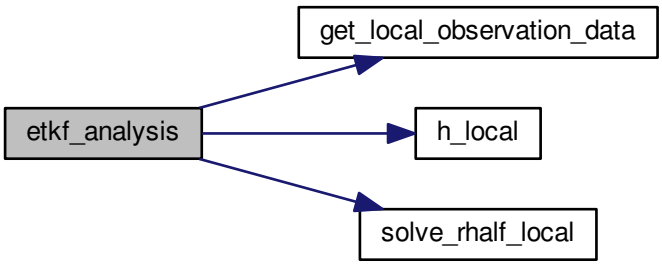
12.48.1 Function/Subroutine Documentation

12.48.1.1 subroutine etkf_analysis (integer, intent(in) *num_hor,* integer, intent(in) *num_ver,* integer, intent(in) *this_hor,* integer, intent(in) *this_ver,* integer, intent(in) *boundary,* real(kind=rk), dimension(statedim,n), intent(inout) *x,* integer, intent(in) *N,* integer, intent(in) *stateDim,* integer, intent(in) *obsDim,* real(kind=rk), intent(in) *rho*)

subroutine to perform the ensemble transform Kalman filter

Definition at line 34 of file etkf_analysis.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

12.49 src/filters/letkf_analysis.f90 File Reference

Functions/Subroutines

- subroutine letkf_analysis

  subroutine to perform the ensemble transform Kalman filter as part of L-ETKF

12.49.1 Function/Subroutine Documentation

12.49.1.1 subroutine letkf_analysis ( )

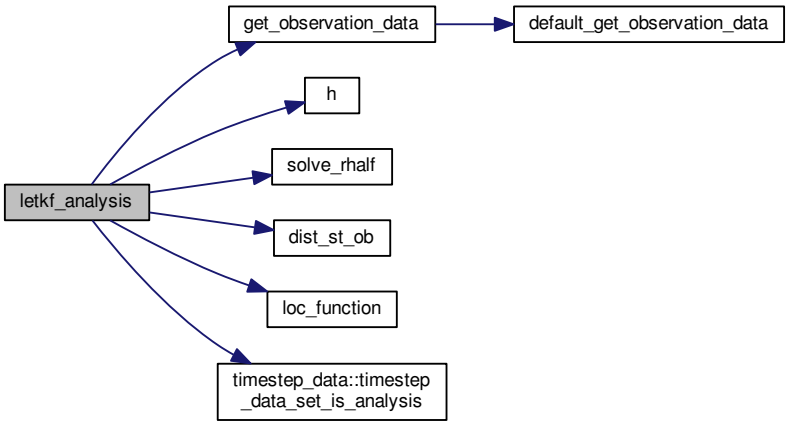
subroutine to perform the ensemble transform Kalman filter as part of L-ETKF

Todo update to allow for non-diagonal R matrices to be used.

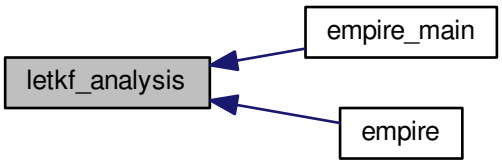
The observation

Definition at line 32 of file letkf_analysis.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.50 src/filters/proposal_filter.f90 File Reference

Functions/Subroutines

- subroutine proposal_filter

  Subroutine to perform nudging in the proposal step of EWPF.

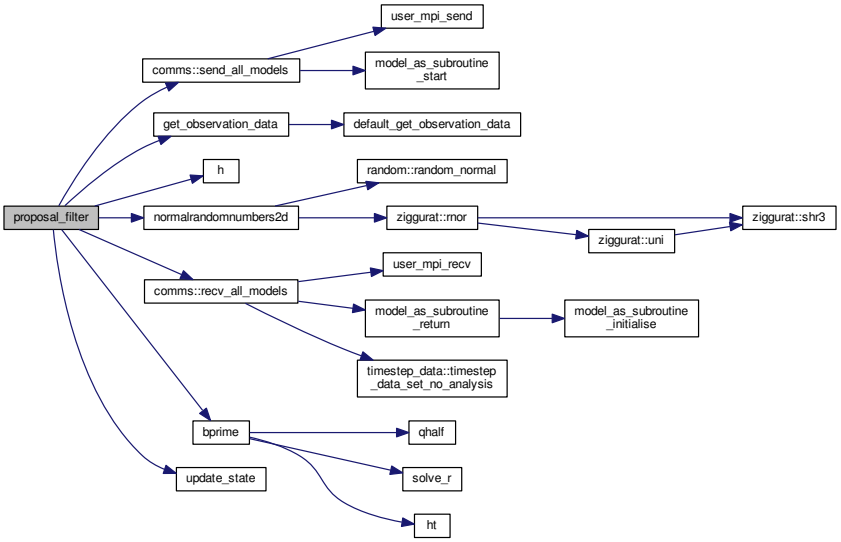
12.50.1 Function/Subroutine Documentation

12.50.1.1 subroutine proposal_filter ( )

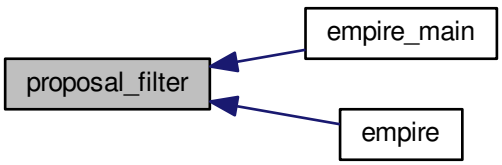
Subroutine to perform nudging in the proposal step of EWPF.

Definition at line 33 of file proposal_filter.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.51 src/filters/sir_filter.f90 File Reference

Functions/Subroutines

- subroutine sir_filter

  Subroutine to perform SIR filter (Sequential Importance Resampling)

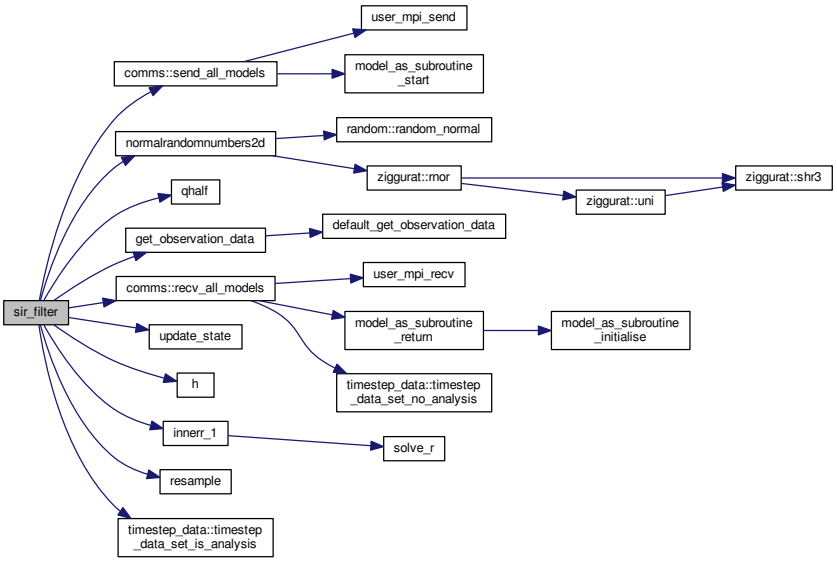
12.51.1 Function/Subroutine Documentation

12.51.1.1 subroutine sir_filter ( )

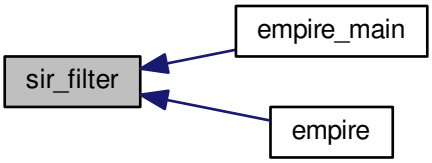
Subroutine to perform SIR filter (Sequential Importance Resampling)

Definition at line 28 of file sir_filter.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.52 src/filters/stochastic_model.f90 File Reference

Functions/Subroutines

- subroutine stochastic_model

  subroutine to simply move the model forward in time one timestep PAB 21-05-2013

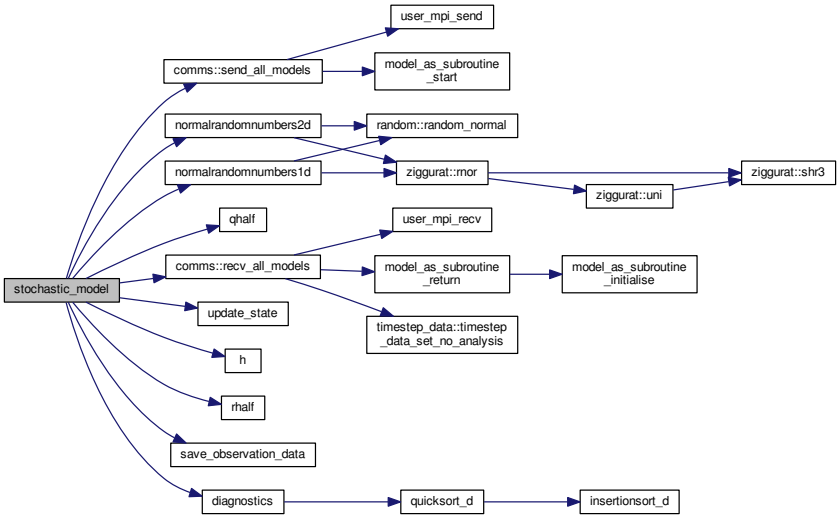
12.52.1 Function/Subroutine Documentation

12.52.1.1 subroutine stochastic_model ( )

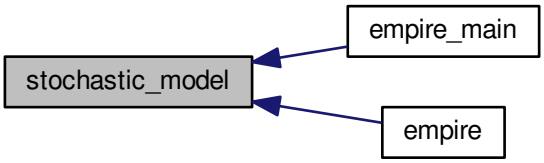
subroutine to simply move the model forward in time one timestep PAB 21-05-2013

Definition at line 33 of file stochastic_model.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.53 src/operations/gen_rand.f90 File Reference

Data Types

- module random_number_controls

Functions/Subroutines

- subroutine uniformrandomnumbers1d (minv, maxv, n, phi)

  generate one dimension of uniform random numbers
- subroutine normalrandomnumbers1d (mean, stdev, n, phi)

generate one dimension of Normal random numbers

- subroutine normalrandomnumbers2d (mean, stdev, n, k, phi)

  generate two dimensional Normal random numbers
- subroutine mixturerandomnumbers1d (mean, stdev, ufac, epsi, n, phi, uniform)

  generate one dimensional vector drawn from mixture density
- subroutine mixturerandomnumbers2d (mean, stdev, ufac, epsi, n, k, phi, uniform)

  generate two dimensional vector, each drawn from mixture density
- subroutine random_seed_mpi (pfid)

  Subroutine to set the random seed across MPI threads.

12.53.1 Function/Subroutine Documentation

12.53.1.1 subroutine mixturerandomnumbers1d (real(kind=kind(1.0d0)), intent(in) *mean,* real(kind=kind(1.0d0)), intent(in) *stdev,* real(kind=kind(1.0d0)), intent(in) *ufac,* real(kind=kind(1.0d0)), intent(in) *epsi,* integer, intent(in) *n,* real(kind=kind(1.0d0)), dimension(n), intent(out) *phi,* logical, intent(out) *uniform*)

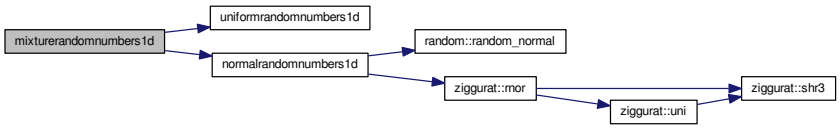
generate one dimensional vector drawn from mixture density

Parameters

in	*mean*	Mean of normal distribution
in	*stdev*	Standard deviation of normal distribution
in	*ufac*	half-width of uniform distribution that is centered on the mean
in	*epsi*	Proportion controlling mixture draw. if random_number $>$ epsi then draw from uniform, else normal
in	*n*	size of output vector
out	*phi*	n dimensional mixture random numbers
out	*uniform*	True if mixture drawn from uniform. False if drawn from normal

Definition at line 155 of file gen_rand.f90.

Here is the call graph for this function:



12.53.1.2 subroutine mixturerandomnumbers2d (real(kind=kind(1.0d0)), intent(in) *mean,* real(kind=kind(1.0d0)), intent(in) *stdev,* real(kind=kind(1.0d0)), intent(in) *ufac,* real(kind=kind(1.0d0)), intent(in) *epsi,* integer, intent(in) *n,* integer, intent(in) *k,* real(kind=kind(1.0d0)), dimension(n,k), intent(out) *phi,* logical, dimension(k), intent(out) *uniform*)

generate two dimensional vector, each drawn from mixture density

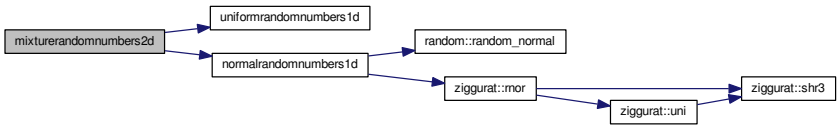
Parameters

in	*mean*	Mean of normal distribution
in	*stdev*	Standard deviation of normal distribution

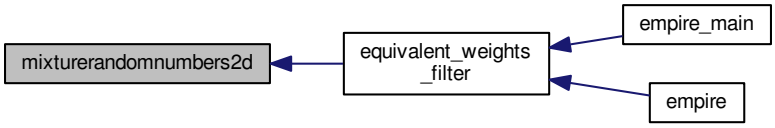
in	*ufac*	half-width of uniform distribution that is centered on the mean
in	*epsi*	Proportion controlling mixture draw. if random_number > epsi then draw from uniform, else normal
in	*n*	first dimension of output vector
in	*k*	second dimension of output vector
out	*phi*	n,k dimensional mixture random numbers
out	*uniform*	k dimensional logical with uniform(i) True if phi(:,i) drawn from uniform. False if drawn from normal

Definition at line 204 of file gen_rand.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.53.1.3 subroutine normalrandomnumbers1d (real(kind=rk), intent(in) *mean,* real(kind=rk), intent(in) *stdev,* integer, intent(in) *n,* real(kind=rk), dimension(n), intent(out) *phi*)

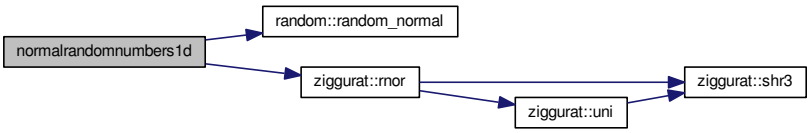
generate one dimension of Normal random numbers

Parameters

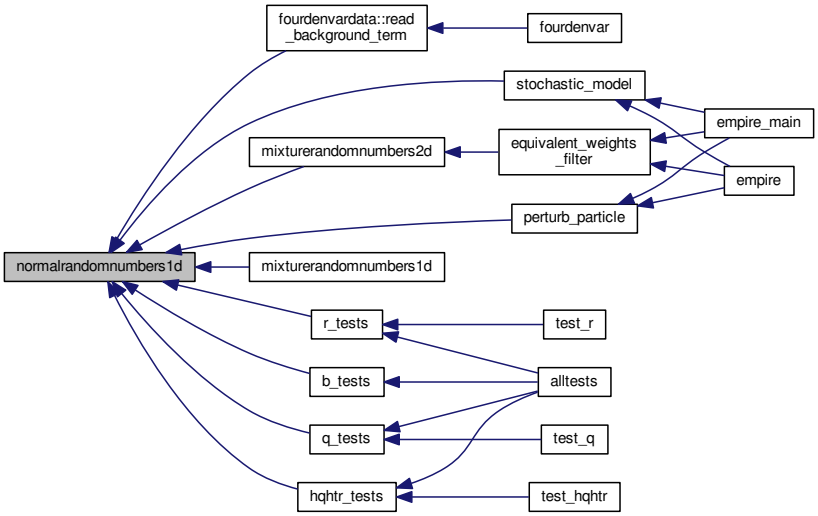
in	*n*	n size of output vector
in	*mean*	mean mean of normal distribution
in	*stdev*	stdev Standard Deviation of normal distribution
out	*phi*	phi n dimensional normal random numbers

Definition at line 73 of file gen_rand.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.53.1.4 subroutine normalrandomnumbers2d (real(kind=rk), intent(in) *mean,* real(kind=rk), intent(in) *stdev,* integer, intent(in) *n,* integer, intent(in) *k,* real(kind=rk), dimension(n,k), intent(out) *phi*)

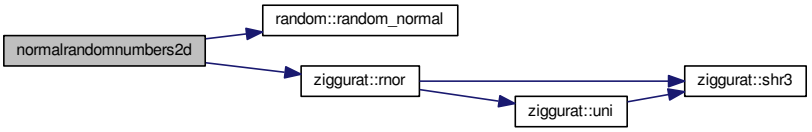
generate two dimensional Normal random numbers

Parameters

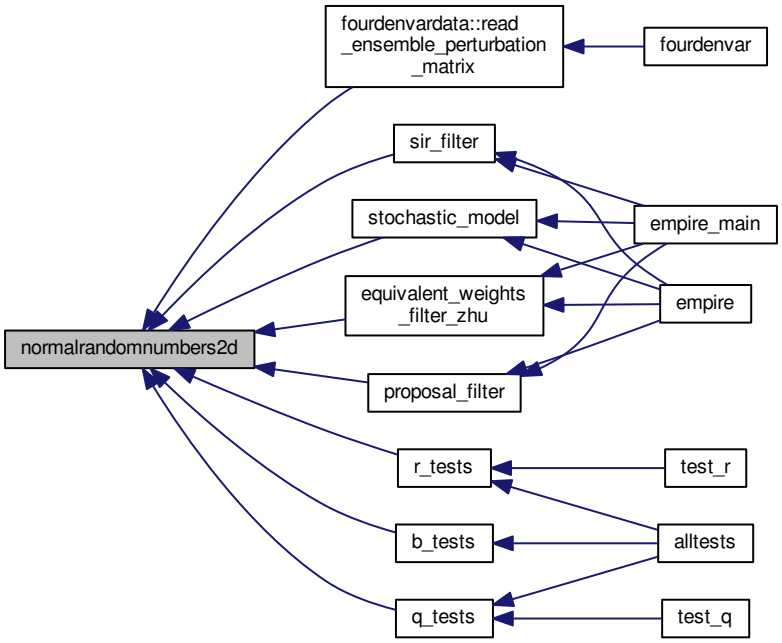
in	*n*	n first dimension of output vector
in	*k*	k second dimension of output vector
in	*mean*	mean mean of normal distribution
in	*stdev*	stdev Standard Deviation of normal distribution
out	*phi*	phi n,k dimensional normal random numbers

Definition at line 107 of file gen_rand.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.53.1.5 subroutine random_seed_mpi (integer, intent(in) *pfid*)

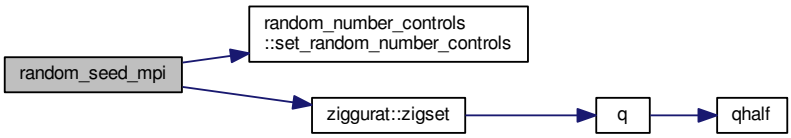
Subroutine to set the random seed across MPI threads.

Parameters

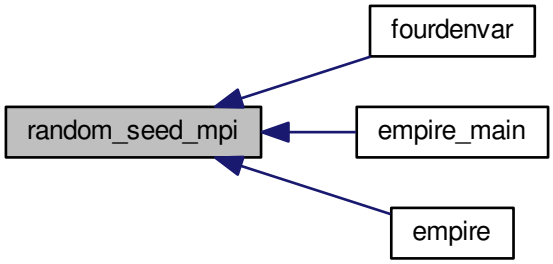
in	*pfid*	The process identifier of the MPI process

Definition at line 230 of file gen_rand.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.53.1.6 subroutine uniformrandomnumbers1d (real(kind=rk), intent(in) *minv,* real(kind=rk), intent(in) *maxv,* integer, intent(in) *n,* real(kind=rk), dimension(n), intent(out) *phi*)

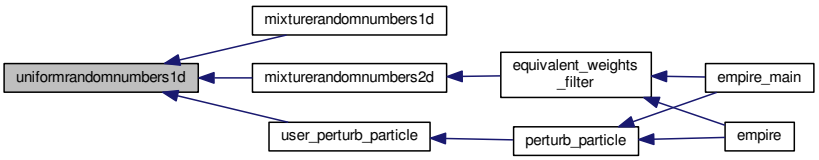
generate one dimension of uniform random numbers

Parameters

in	*n*	n size of output vector
in	*minv*	minv minimum value of uniform distribution
in	*maxv*	maxv maximum value of uniform distribution
out	*phi*	phi n dimensional uniform random numbers

Definition at line 58 of file gen_rand.f90.

Here is the caller graph for this function:



12.54 src/operations/inner_products.f90 File Reference

Functions/Subroutines

- subroutine innerr_1 (n, c, y, w, t)

  subroutine to compute the inner product with R^{-1}
- subroutine innerhqht_plus_r_1 (y, w, t)

  subroutine to compute the inner product with $(HQH^T + R)^{-1}$

12.54.1 Function/Subroutine Documentation

12.54.1.1 subroutine innerhqht_plus_r_1 (real(kind=rk), dimension(obs_dim), intent(in) *y*, real(kind=rk), intent(out) *w*, integer, intent(in) *t*)

subroutine to compute the inner product with $(HQH^T + R)^{-1}$

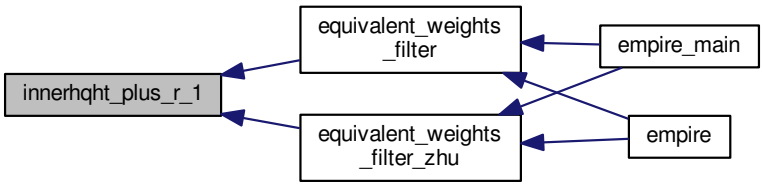
Parameters

in	*y*	vector in observation space
out	*w*	scalar with value $y^T R^{-1} y$
in	*t*	current timestep

Definition at line 76 of file inner_products.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



12.54.1.2 subroutine innerr_1 (integer, intent(in) *n*, integer, intent(in) *c*, real(kind=rk), dimension(n,c), intent(in) *y*, real(kind=rk), dimension(c), intent(out) *w*, integer, intent(in) *t*)

subroutine to compute the inner product with R^{-1}

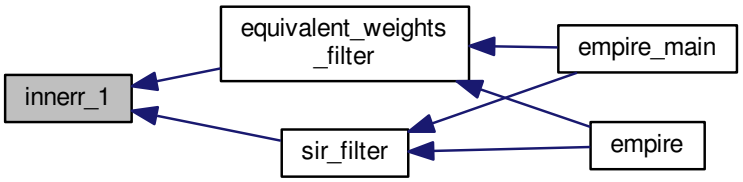
Parameters

in	*n*	length of each vector in y
in	*c*	number of vectors in y
in	*y*	multiple vectors in observation space (pf%count of them)
out	*w*	multiple scalars (pf%count) where w(i) has the value $y(:,i)^T R^{-1} y(:,i)$
in	*t*	current timestep

Definition at line 36 of file inner_products.f90.

Here is the call graph for this function:

Here is the caller graph for this function:



12.55 src/operations/operator_wrappers.f90 File Reference

Functions/Subroutines

- subroutine k (y, x)

  Subroutine to apply K to a vector y in observation space where $K := QH^T(HQH^T + R)^{-1}$.
- subroutine bprime (y, x, QHtR_1y, normaln, betan)

  subroutine to calculate nudging term and correlated random errors efficiently

12.55.1 Function/Subroutine Documentation

12.55.1.1 subroutine bprime (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y,* real(kind=rk), dimension(state_dim,pf%count), intent(out) *x,* real(kind=rk), dimension(state_dim,pf%count), intent(out) *QHtR_1y,* real(kind=rk), dimension(state_dim,pf%count), intent(in) *normaln,* real(kind=rk), dimension(state_dim,pf%count), intent(out) *betan*)

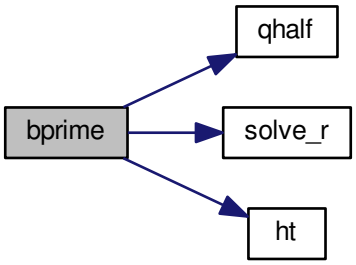
subroutine to calculate nudging term and correlated random errors efficiently

Parameters

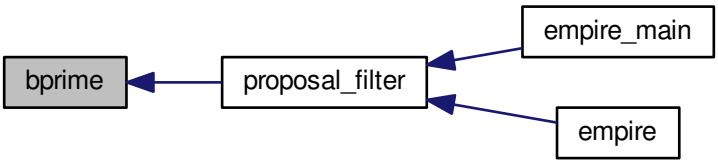
in	*y*	(obs_dim,pf%count) vectors of innovations $y - H(x^{n-1})$
out	*x*	(state_dim,pf%count) vectors of $\rho H^T R^{-1}[y - H(x^{n-1})]$
out	*QHtR_1y*	(state_dim,pf%count) vectors of $\rho Q H^T R^{-1}[y - H(x^{n-1})]$
in	*normaln*	(state_dim,pf%count) uncorrelated random vectors such that normaln(:,i) $\sim \mathcal{N}(0, I)$
out	*betan*	(state_dim,pf%count) correlated random vectors such that betan(:,i) $\sim \mathcal{N}(0, Q)$

Definition at line 109 of file operator_wrappers.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.55.1.2 subroutine k (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y*, real(kind=rk), dimension(state_dim,pf%count), intent(out) *x*)

Subroutine to apply K to a vector y in observation space where $K := QH^T(HQH^T + R)^{-1}$.

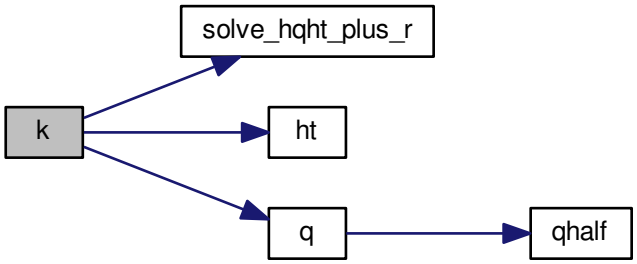
Parameters

in	*y*	vector in observation space

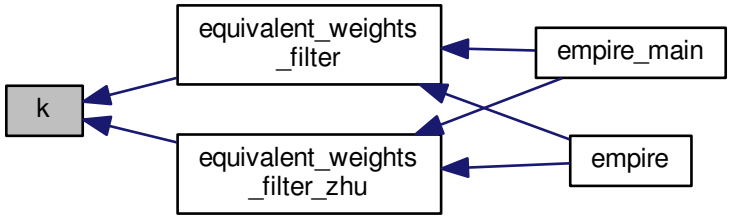
out	x	vector in state space

Definition at line 32 of file operator_wrappers.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.56 src/operations/perturb_particle.f90 File Reference

Functions/Subroutines

- subroutine perturb_particle (x)

  Subroutine to perturb state vector governed by the init option.

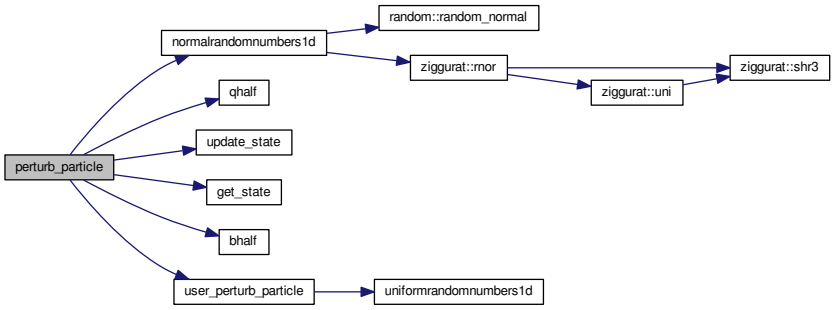
12.56.1 Function/Subroutine Documentation

12.56.1.1 subroutine perturb_particle (real(kind=rk), dimension(state_dim), intent(inout) *x*)

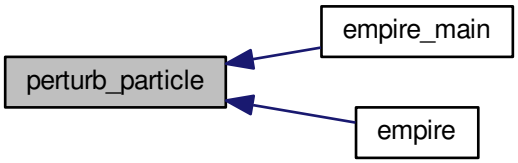
Subroutine to perturb state vector governed by the init option.

Definition at line 30 of file perturb_particle.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.57 src/operations/phalf.f90 File Reference

Functions/Subroutines

- subroutine phalf (nrhs, x, Px)

  subroutine to take a full state vector x and return $P^{1/2}x$ in state space.

12.57.1 Function/Subroutine Documentation

12.57.1.1 subroutine phalf (integer, intent(in) *nrhs*, real(kind=rk), dimension(state_dim,nrhs), intent(in) *x*, real(kind=rk), dimension(state_dim,nrhs), intent(out) *Px*)

subroutine to take a full state vector x and return $P^{1/2}x$ in state space.

Given x compute $P^{\frac{1}{2}}x$

where $P = (Q^{-1} + H^T R^{-1} H)^{-1} = Q^{\frac{1}{2}}(I + Q^{\frac{1}{2}} H^T R^{-1} H Q^{\frac{1}{2}})^{-1} Q^{\frac{1}{2}}$

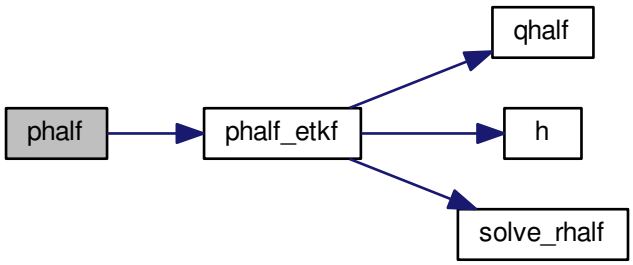
This is required for the Zhu Equal weights particle filter equivalent_weights_filter_zhu

Parameters

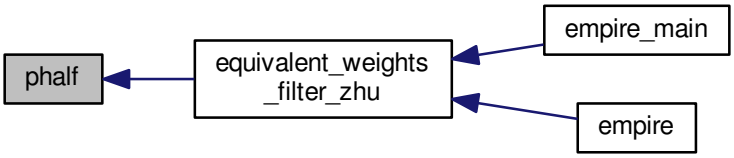
in	*nrhs*	the number of right hand sides
in	*x*	the input vector
out	*px*	the resulting vector where Px $= P^{\frac{1}{2}}x$

Definition at line 11 of file phalf.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.58 src/operations/phalf_etkf.f90 File Reference

Functions/Subroutines

- subroutine phalf_etkf (nrhs, x, px)

  Subroutine to go from N(0,Q) to N(0,P)

12.58.1 Function/Subroutine Documentation

12.58.1.1 subroutine phalf_etkf (integer, intent(in) *nrhs*, real(kind=rk), dimension(state_dim,nrhs), intent(in) *x*, real(kind=rk), dimension(state_dim,nrhs), intent(out) *px*)

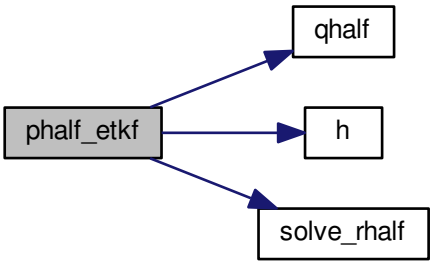
Subroutine to go from N(0,Q) to N(0,P)

Parameters

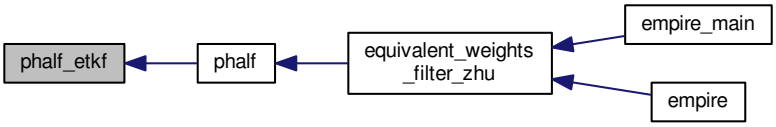
in	*nrhs*	the number of right hand sides
in	*x*	the input vector, assumed to be columns N(0,I)
out	*px*	the resulting vector where Px $= P^{\frac{1}{2}}x$

Definition at line 30 of file phalf_etkf.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.59 src/operations/resample.f90 File Reference

Functions/Subroutines

- subroutine resample

  Subroutine to perform Universal Importance Resampling.

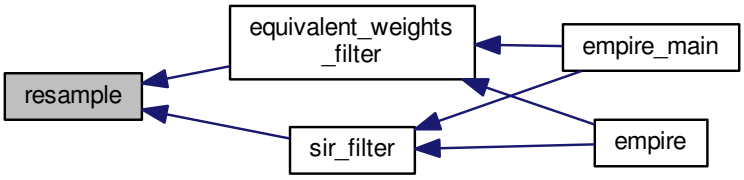
12.59.1 Function/Subroutine Documentation

12.59.1.1 subroutine resample ( )

Subroutine to perform Universal Importance Resampling.

Definition at line 28 of file resample.f90.

Here is the caller graph for this function:



12.60 src/operations/update_state.f90 File Reference

Functions/Subroutines

- subroutine update_state (state, fpsi, kgain, betan)

  Subroutine to update the state.

12.60.1 Function/Subroutine Documentation

12.60.1.1 subroutine update_state (real(kind=rk), dimension(state_dim), intent(out) *state,* real(kind=rk), dimension(state_dim), intent(in) *fpsi,* real(kind=rk), dimension(state_dim), intent(in) *kgain,* real(kind=rk), dimension(state_dim), intent(inout) *betan*)

Subroutine to update the state.

This subroutine is here because, mathematically, in a particle filter $x^{k+1} = f(x^k) + A^k + \xi^k$

However sometimes the result needs to be bounded, some variables need to be exactly related or maybe even something else.

This can be changed for the specific model if it needs to be, in order to bound variables etc.

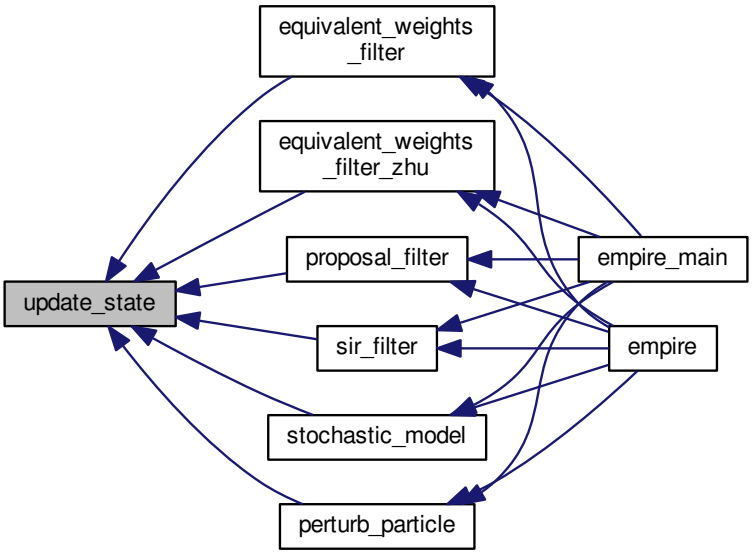
NOTE this the theory not mathematically correct.

Parameters

in	*fpsi*	deterministic model update $f(x^{n-1})$
in	*kgain*	nudging term
in,out	*betan*	Stochastic term
out	*state*	The updated state vector

Definition at line 47 of file update_state.f90.

Here is the caller graph for this function:



12.61 src/optim/CG+/call.f90 File Reference

Functions/Subroutines

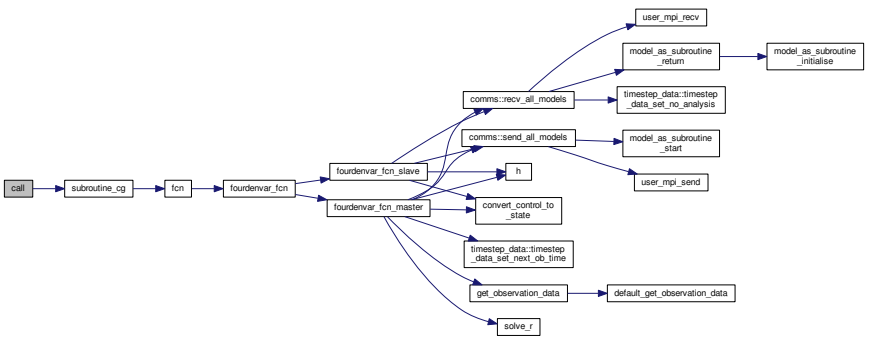
- program call

12.61.1 Function/Subroutine Documentation

12.61.1.1 program call ( )

Definition at line 1 of file call.f90.

Here is the call graph for this function:



12.62 src/optim/CG+/MPI/call.f90 File Reference

Functions/Subroutines

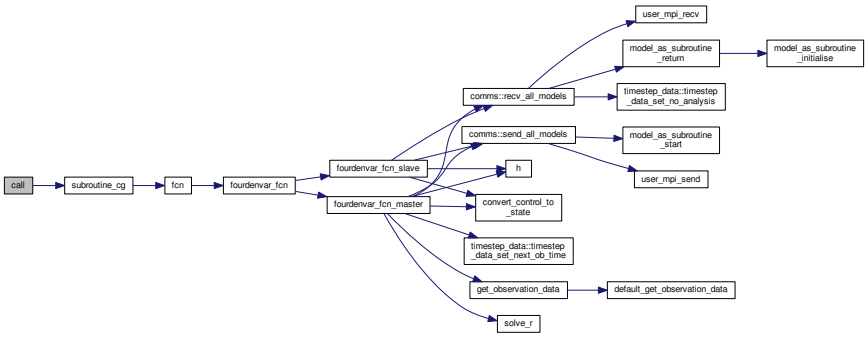
- program call

12.62.1 Function/Subroutine Documentation

12.62.1.1 program call ( )

Definition at line 1 of file call.f90.

Here is the call graph for this function:



12.63 src/optim/Lbfgsb.3.0/call.f90 File Reference

Functions/Subroutines

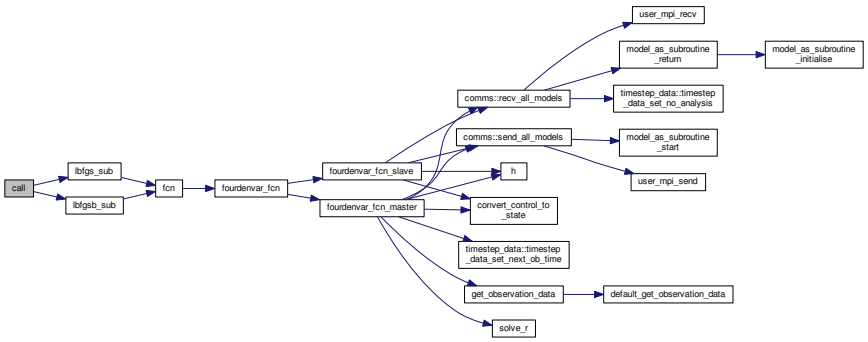
- program call

12.63.1 Function/Subroutine Documentation

12.63.1.1 program call ( )

Definition at line 1 of file call.f90.

Here is the call graph for this function:



12.64 src/optim/CG+/cgsub.f90 File Reference

Functions/Subroutines

- subroutine subroutine_cg (method, n, epsin, x)

  Nonlinear Conjugate gradient method as callable subroutine.

12.64.1 Function/Subroutine Documentation

12.64.1.1 subroutine subroutine_cg (integer, intent(in) *method,* integer, intent(in) *n,* real(kind=rk), intent(in) *epsin,* real(kind=rk), dimension(n), intent(inout) *x*)

Nonlinear Conjugate gradient method as callable subroutine.

Main program for running the conjugate gradient methods described in the paper:

Gilbert, J.C. and Nocedal, J. (1992). "Global Convergence Properties of Conjugate Gradient Methods", SIAM Journal on Optimization, Vol. 2, pp. 21-42.

A web-based Server which solves unconstrained nonlinear optimization problems using this Conjugate Gradient code can be found at:

http://www-neos.mcs.anl.gov/neos/solvers/UCO:CGPLUS/

Written by G. Liu, J. Nocedal and R. Waltz October 1998

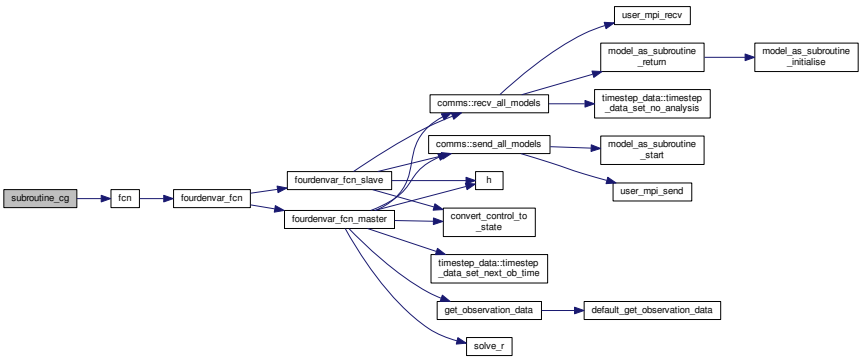
modified to be a callable subroutine by Philip A Browne Jan 2015

Parameters

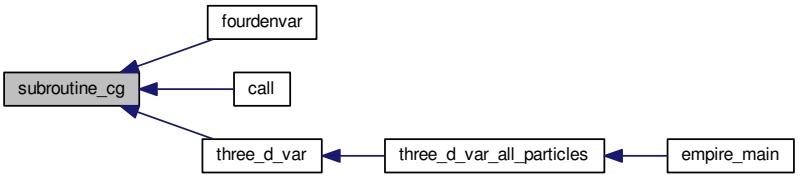
in	*method*	which CG method to use
in	*n*	the dimension of the state vector
in	*epsin*	the value of EPS to be used as convergence tolerance
in,out	*x*	on entry the initial guess, on exit is the optimized state vector

Definition at line 27 of file cgsub.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.65 src/optim/CG+/MPI/cgsub.f90 File Reference

Functions/Subroutines

- subroutine subroutine_cg (method, n, epsin, x, mpi_comm, mpi_size)

  Nonlinear Conjugate gradient method as callable subroutine.

12.65.1 Function/Subroutine Documentation

12.65.1.1 subroutine subroutine_cg (integer, intent(in) *method,* integer, intent(in) *n,* real(kind=rk), intent(in) *epsin,* real(kind=rk), dimension(n), intent(inout) *x,* integer, intent(in) *mpi_comm,* integer, intent(in) *mpi_size*)

Nonlinear Conjugate gradient method as callable subroutine.

Main program for running the conjugate gradient methods described in the paper:

Gilbert, J.C. and Nocedal, J. (1992). "Global Convergence Properties of Conjugate Gradient Methods", SIAM Journal on Optimization, Vol. 2, pp. 21-42.

A web-based Server which solves unconstrained nonlinear optimization problems using this Conjugate Gradient code can be found at:

http://www-neos.mcs.anl.gov/neos/solvers/UCO:CGPLUS/

Written by G. Liu, J. Nocedal and R. Waltz October 1998

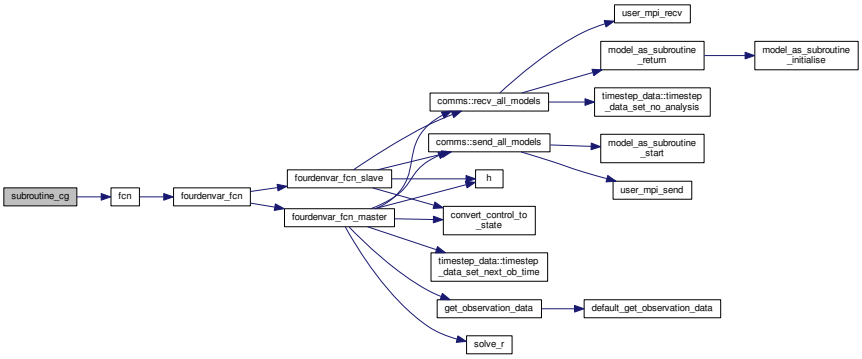
modified to be a callable subroutine by Philip A Browne Jan 2015

Parameters

in	*method*	which CG method to use
in	*n*	the dimension of the state vector
in	*epsin*	the value of EPS to be used as convergence tolerance
in,out	*x*	on entry the initial guess, on exit is the optimized state vector

Definition at line 27 of file cgsub.f90.

Here is the call graph for this function:



12.66 src/optim/CG+/fcn.f90 File Reference

Functions/Subroutines

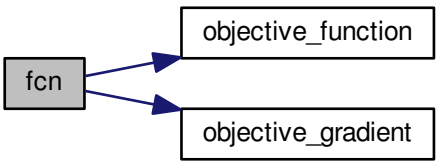
- subroutine fcn (n, x, f, g)

12.66.1 Function/Subroutine Documentation

12.66.1.1 subroutine fcn (integer *n*, real(kind=kind(1.0d0)), dimension(n), intent(in) *x*, real(kind=kind(1.0d0)), intent(out) *f*, real(kind=kind(1.0d0)), dimension(n), intent(out) *g*)

Definition at line 1 of file fcn.f90.

Here is the call graph for this function:



12.67 src/optim/CG+/MPI/fcn.f90 File Reference

Functions/Subroutines

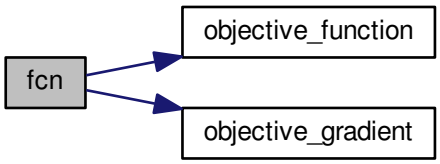
- subroutine fcn (n, x, f, g)

12.67.1 Function/Subroutine Documentation

12.67.1.1 subroutine fcn (integer *n*, real(kind=kind(1.0d0)), dimension(n), intent(in) *x*, real(kind=kind(1.0d0)), intent(out) *f*, real(kind=kind(1.0d0)), dimension(n), intent(out) *g*)

Definition at line 1 of file fcn.f90.

Here is the call graph for this function:



12.68 src/optim/Lbfgsb.3.0/fcn.f90 File Reference

Functions/Subroutines

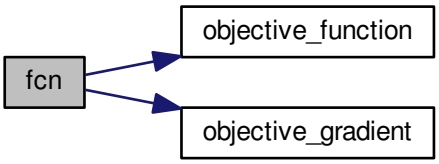
- subroutine fcn (n, x, f, g)

12.68.1 Function/Subroutine Documentation

12.68.1.1 subroutine fcn (integer *n*, real(kind=kind(1.0d0)), dimension(n), intent(in) *x*, real(kind=kind(1.0d0)), intent(out) *f*, real(kind=kind(1.0d0)), dimension(n), intent(out) *g*)

Definition at line 1 of file fcn.f90.

Here is the call graph for this function:



12.69 src/var/fcn.f90 File Reference

Functions/Subroutines

- subroutine fcn (n, x, f, g)

  This is the subroutine which the optimization routines call to get the objective function value and its gradient.

12.69.1 Function/Subroutine Documentation

12.69.1.1 subroutine fcn (integer, intent(in) *n*, real(kind=kind(1.0d0)), dimension(n), intent(in) *x*, real(kind=kind(1.0d0)), intent(out) *f*, real(kind=kind(1.0d0)), dimension(n), intent(out) *g*)

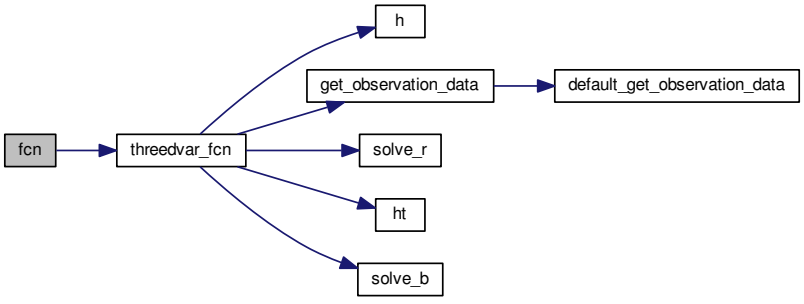
This is the subroutine which the optimization routines call to get the objective function value and its gradient.

Parameters

in	*n*	the dimension of the optimzation problem
in	*x*	the current optimization state
out	*f*	the objective function value
out	*g*	the gradient of the objective function

Definition at line 30 of file fcn.f90.

Here is the call graph for this function:



12.70 src/optim/CG+/MPI/objective_function.f90 File Reference

Functions/Subroutines

- subroutine objective_function (n, x, f)

12.70.1 Function/Subroutine Documentation

12.70.1.1 subroutine objective_function (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(in) *x,* real(kind=rk), intent(out) *f*)

Definition at line 1 of file objective_function.f90.

Here is the caller graph for this function:

12.71 src/optim/CG+/objective_function.f90 File Reference

Functions/Subroutines

- subroutine objective_function (n, x, f)

12.71.1 Function/Subroutine Documentation

12.71.1.1 subroutine objective_function (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(in) *x,* real(kind=rk), intent(out) *f*)

Definition at line 1 of file objective_function.f90.

12.72 src/optim/Lbfgsb.3.0/objective_function.f90 File Reference

Functions/Subroutines

- subroutine objective_function (n, x, f)

12.72.1 Function/Subroutine Documentation

12.72.1.1 subroutine objective_function (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(in) *x,* real(kind=rk), intent(out) *f*)

Definition at line 1 of file objective_function.f90.

12.73 src/optim/CG+/MPI/objective_gradient.f90 File Reference

Functions/Subroutines

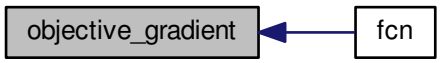
- subroutine objective_gradient (n, x, g)

12.73.1 Function/Subroutine Documentation

12.73.1.1 subroutine objective_gradient (integer, intent(in) *n*, real(kind=rk), dimension(n), intent(in) *x*, real(kind=rk), dimension(n), intent(out) *g*)

Definition at line 1 of file objective_gradient.f90.

Here is the caller graph for this function:



12.74 src/optim/CG+/objective_gradient.f90 File Reference

Functions/Subroutines

- subroutine objective_gradient (n, x, g)

12.74.1 Function/Subroutine Documentation

12.74.1.1 subroutine objective_gradient (integer, intent(in) *n*, real(kind=rk), dimension(n), intent(in) *x*, real(kind=rk), dimension(n), intent(out) *g*)

Definition at line 1 of file objective_gradient.f90.

12.75 src/optim/Lbfgsb.3.0/objective_gradient.f90 File Reference

Functions/Subroutines

- subroutine objective_gradient (n, x, g)

12.75.1 Function/Subroutine Documentation

12.75.1.1 subroutine objective_gradient (integer, intent(in) *n*, real(kind=rk), dimension(n), intent(in) *x*, real(kind=rk), dimension(n), intent(out) *g*)

Definition at line 1 of file objective_gradient.f90.

12.76 src/optim/CG+/MPI/README.txt File Reference

12.77 src/optim/Lbfgsb.3.0/driver1.f90 File Reference

Functions/Subroutines

- program driver

12.77.1 Function/Subroutine Documentation

12.77.1.1 program driver ( )

Definition at line 190 of file driver1.f90.

12.78 src/optim/Lbfgsb.3.0/driver2.f90 File Reference

Functions/Subroutines

- program driver

12.78.1 Function/Subroutine Documentation

12.78.1.1 program driver ( )

Definition at line 46 of file driver2.f90.

12.79 src/optim/Lbfgsb.3.0/driver3.f90 File Reference

Functions/Subroutines

- program driver

12.79.1 Function/Subroutine Documentation

12.79.1.1 program driver ( )

Definition at line 47 of file driver3.f90.

12.80 src/optim/Lbfgsb.3.0/lbfgs_sub.f90 File Reference

Functions/Subroutines

- subroutine lbfgs_sub (n, factr_in, pgtol_in, x)

  Limited memory BFGS unconstrained optimization code as callable subroutine.

12.80.1 Function/Subroutine Documentation

12.80.1.1 subroutine lbfgs_sub (integer, intent(in) *n,* real(kind=dp), intent(in) *factr_in,* real(kind=dp), intent(in) *pgtol_in,* real(kind=dp), dimension(n), intent(inout) *x*)

Limited memory BFGS unconstrained optimization code as callable subroutine.

L-BFGS-B is a code for solving large nonlinear optimization problems with simple bounds on the variables.

The code can also be used for unconstrained problems and is as efficient for these problems as the earlier limited memory code L-BFGS.

This is the simplest driver in the package. It uses all the default settings of the code.

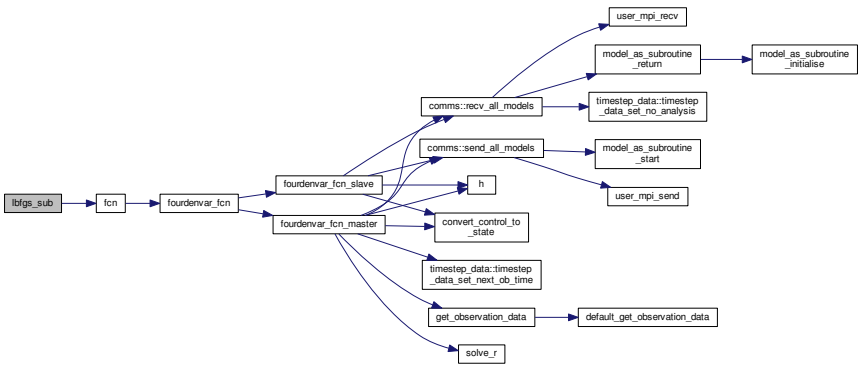
```
References:

    [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, ``A limited
    memory algorithm for bound constrained optimization'',
    SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.

    [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, ``L-BFGS-B: FORTRAN
    Subroutines for Large Scale Bound Constrained Optimization''
    Tech. Report, NAM-11, EECS Department, Northwestern University,
    1994.


      (Postscript files of these papers are available via anonymous
       ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)

                           *  *  *

     March 2011   (latest revision)
     Optimization Center at Northwestern University
     Instituto Tecnologico Autonomo de Mexico

     Jorge Nocedal and Jose Luis Morales
```

Parameters

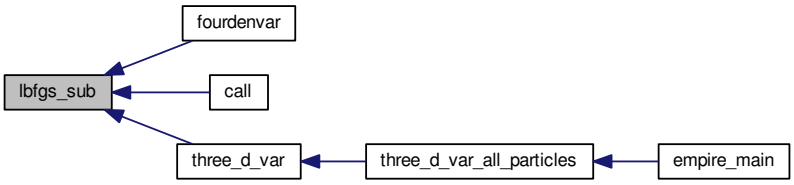
in	*n*	the size of the state vector
in	*factr_in*	the factr tolerance in the stopping criteria
in	*pgtol_in*	the pgtol tolerance in the stopping criteria
in,out	*x*	on entry the initial guess, on exit the optimized state vector

Definition at line 198 of file lbfgs_sub.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.81 src/optim/Lbfgsb.3.0/lbfgsb_sub.f90 File Reference

Functions/Subroutines

- subroutine lbfgsb_sub (n, factr_in, pgtol_in, x, nbd, l, u)

  Limited memory BFGS bound constrained optimization code as callable subroutine.

12.81.1 Function/Subroutine Documentation

12.81.1.1 subroutine lbfgsb_sub (integer, intent(in) *n,* real(kind=dp), intent(in) *factr_in,* real(kind=dp), intent(in) *pgtol_in,* real(kind=dp), dimension(n), intent(inout) *x,* integer, dimension(n), intent(in) *nbd,* real(kind=dp), dimension(n), intent(in) *l,* real(kind=dp), dimension(n), intent(in) *u*)

Limited memory BFGS bound constrained optimization code as callable subroutine.

L-BFGS-B is a code for solving large nonlinear optimization problems with simple bounds on the variables.

The code can also be used for unconstrained problems and is as efficient for these problems as the earlier limited memory code L-BFGS.

This is the simplest driver in the package. It uses all the default settings of the code.

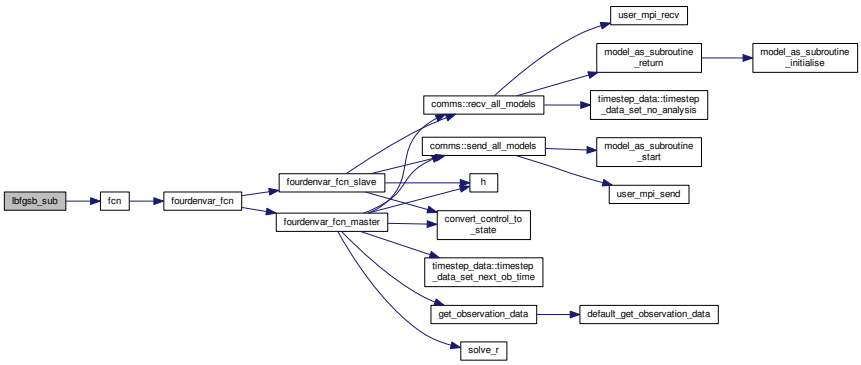
```
 References:

    [1] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, ``A limited
    memory algorithm for bound constrained optimization'',
    SIAM J. Scientific Computing 16 (1995), no. 5, pp. 1190--1208.

    [2] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, ``L-BFGS-B: FORTRAN
    Subroutines for Large Scale Bound Constrained Optimization''
    Tech. Report, NAM-11, EECS Department, Northwestern University,
    1994.


      (Postscript files of these papers are available via anonymous
       ftp to eecs.nwu.edu in the directory pub/lbfgs/lbfgs_bcm.)

                           *  *  *

     March 2011   (latest revision)
     Optimization Center at Northwestern University
     Instituto Tecnologico Autonomo de Mexico

     Jorge Nocedal and Jose Luis Morales
```

Parameters

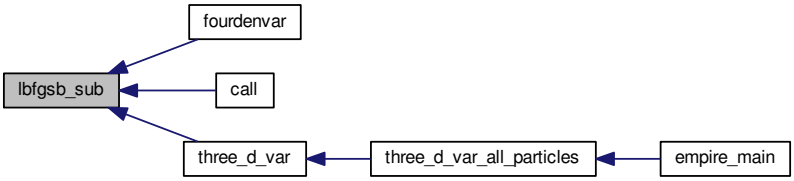
in	*n*	the size of the state vector
in	*factr_in*	the factr tolerance in the stopping criteria
in	*pgtol_in*	the pgtol tolerance in the stopping criteria
in,out	*x*	on entry the initial guess, on exit the optimized state vector
in	*nbd*	nbd is an INTEGER array of dimension n that must be set by the user to the type of bounds imposed on the variables: nbd(i)=0 if x(i) is unbounded, 1 if x(i) has only a lower bound, 2 if x(i) has both lower and upper bounds, 3 if x(i) has only an upper bound.
in	*l*	l is a DOUBLE PRECISION array of length n that must be set by the user to the values of the lower bounds on the variables. If the i-th variable has no lower bound, l(i) need not be defined.
in	*u*	u is a DOUBLE PRECISION array of length n that must be set by the user to the values of the upper bounds on the variables. If the i-th variable has no upper bound, u(i) need not be defined.

Definition at line 210 of file lbfgsb_sub.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.82 src/optim/Lbfgsb.3.0/License.txt File Reference

Functions

- clause license ("New BSD License"or"Modified BSD License") New BSD License Author Regents of the University of California Publisher Public Domain Published July 22

- OR BUSINESS INTERRUPTION HOWEVER CAUSED AND ON ANY THEORY OF WHETHER IN STRICT OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE

Variables

- clause DFSG compatible Yes[7] FSF approved Yes[1] OSI approved Yes[3] GPL compatible Yes[1] Copyleft No[1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on July
- clause DFSG compatible Yes[7] FSF approved Yes[1] OSI approved Yes[3] GPL compatible Yes[1] Copyleft No[1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on by William Hoskins
- clause DFSG compatible Yes[7] FSF approved Yes[1] OSI approved Yes[3] GPL compatible Yes[1] Copyleft No[1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on by William Director of the Office of Technology Licensing for UC Berkeley[8] Other BSD distributions removed the clause
- clause DFSG compatible Yes[7] FSF approved Yes[1] OSI approved Yes[3] GPL compatible Yes[1] Copyleft No[1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on by William Director of the Office of Technology Licensing for UC Berkeley[8] Other BSD distributions removed the but many similar clauses remain in BSD derived code from other sources
- LOSS OF USE
- LOSS OF DATA
- LOSS OF OR PROFITS
- OR BUSINESS INTERRUPTION HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY
- OR BUSINESS INTERRUPTION HOWEVER CAUSED AND ON ANY THEORY OF WHETHER IN CONT↩ RACT

12.82.1 Function Documentation

12.82.1.1 clause license ("New BSD License"or"Modified BSD License")

12.82.1.2 OR BUSINESS INTERRUPTION HOWEVER CAUSED AND ON ANY THEORY OF WHETHER IN STRICT OR TORT (INCLUDING NEGLIGENCE OR *OTHERWISE*)

12.82.2 Variable Documentation

12.82.2.1 clause DFSG compatible Yes [7] FSF approved Yes [1] OSI approved Yes [3] GPL compatible Yes [1] Copyleft No [1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on by William Director of the Office of Technology Licensing for UC Berkeley [8] Other BSD distributions removed the clause

Definition at line 14 of file License.txt.

12.82.2.2 OR BUSINESS INTERRUPTION HOWEVER CAUSED AND ON ANY THEORY OF WHETHER IN CONTRACT

Definition at line 50 of file License.txt.

12.82.2.3 LOSS OF DATA

Definition at line 49 of file License.txt.

12.82.2.4 clause DFSG compatible Yes [7] FSF approved Yes [1] OSI approved Yes [3] GPL compatible Yes [1] Copyleft No [1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on by William Hoskins

Definition at line 14 of file License.txt.

12.82.2.5 clause DFSG compatible Yes [7] FSF approved Yes [1] OSI approved Yes [3] GPL compatible Yes [1] Copyleft No [1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on July

Definition at line 14 of file License.txt.

12.82.2.6 OR BUSINESS INTERRUPTION HOWEVER CAUSED AND ON ANY THEORY OF WHETHER IN STRICT LIABILITY

Definition at line 50 of file License.txt.

12.82.2.7 LOSS OF OR PROFITS

Definition at line 49 of file License.txt.

12.82.2.8 clause DFSG compatible Yes [7] FSF approved Yes [1] OSI approved Yes [3] GPL compatible Yes [1] Copyleft No [1] Copyfree Yes Linking from code with a different license Yes The advertising clause was removed from the license text in the official BSD on by William Director of the Office of Technology Licensing for UC Berkeley [8] Other BSD distributions removed the but many similar clauses remain in BSD derived code from other sources

Definition at line 14 of file License.txt.

12.82.2.9 LOSS OF USE

Definition at line 49 of file License.txt.

12.83 src/smoothers/letks.f90 File Reference

Data Types

- module letks_data

  module for doing things related to the LETKS:
- type letks_data::letks_local

12.84 src/tests/alltests.f90 File Reference

Functions/Subroutines

- program alltests

  program to run all tests of user specific functions

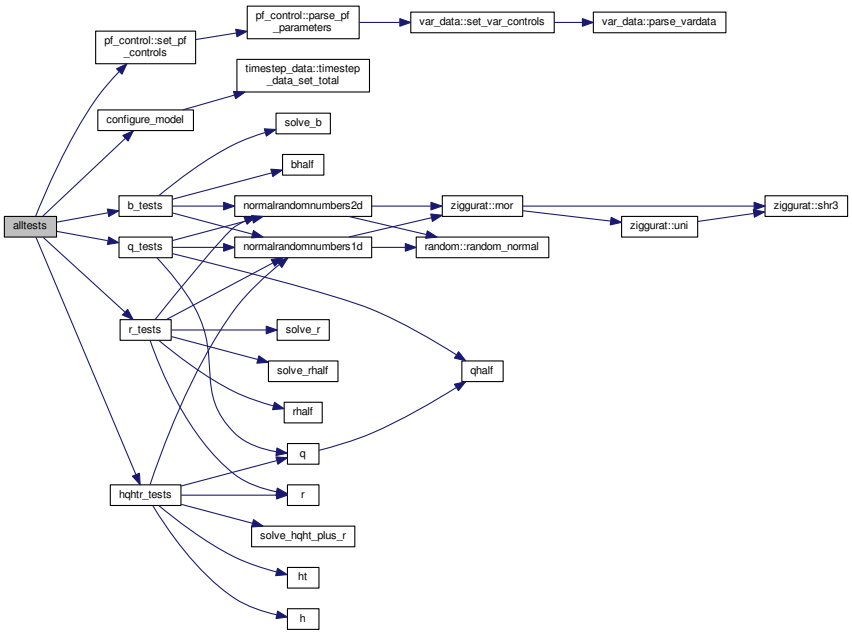
12.84.1 Function/Subroutine Documentation

12.84.1.1 program alltests ( )

program to run all tests of user specific functions

Definition at line 31 of file alltests.f90.

Here is the call graph for this function:



12.85 src/tests/test_h.f90 File Reference

12.86 src/tests/test_hqhtr.f90 File Reference

Functions/Subroutines

- program test_hqhtr

  program to run tests of user supplied linear solve

12.86.1 Function/Subroutine Documentation

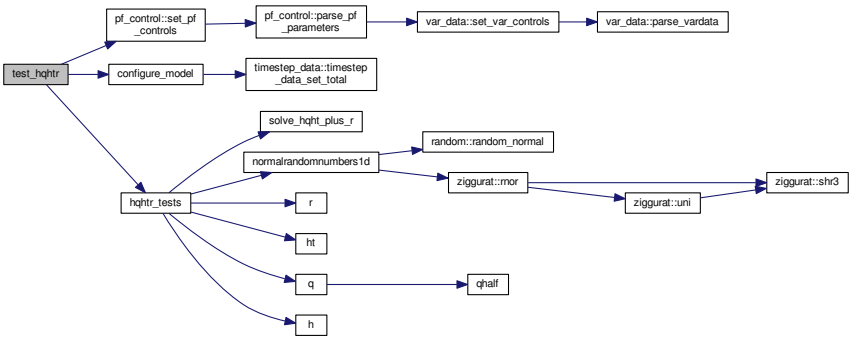
12.86.1.1 program test_hqhtr ( )

program to run tests of user supplied linear solve

$(HQH^T + R)^{-1}$

Definition at line 33 of file test_hqhtr.f90.

Here is the call graph for this function:



12.87 src/tests/test_q.f90 File Reference

Functions/Subroutines

- program test_q

  program to run tests of user supplied model error covariance matrix

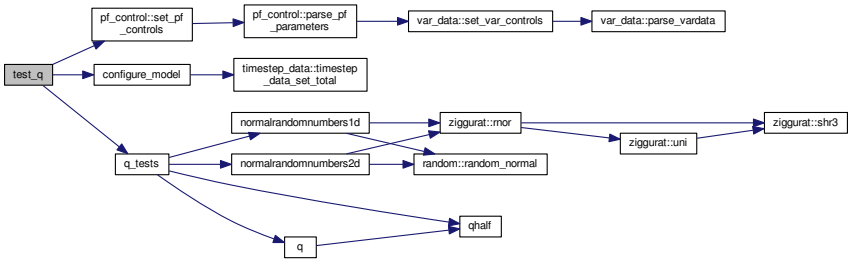
12.87.1 Function/Subroutine Documentation

12.87.1.1 program test_q ( )

program to run tests of user supplied model error covariance matrix

Definition at line 31 of file test_q.f90.

Here is the call graph for this function:



12.88 src/tests/test_r.f90 File Reference

Functions/Subroutines

- program test_r

  program to run all tests of user supplied observation error covariance matrix/

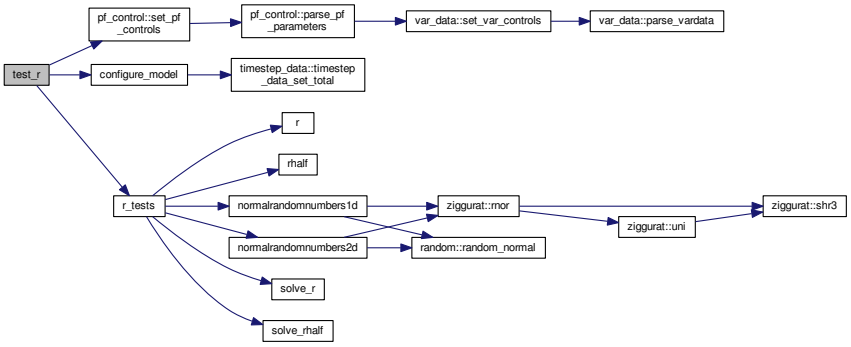
12.88.1 Function/Subroutine Documentation

12.88.1.1 program test_r ( )

program to run all tests of user supplied observation error covariance matrix/

Definition at line 31 of file test_r.f90.

Here is the call graph for this function:



12.89 src/tests/tests.f90 File Reference

Functions/Subroutines

- subroutine r_tests ()
- subroutine q_tests ()
- subroutine hqhtr_tests ()
- subroutine b_tests ()

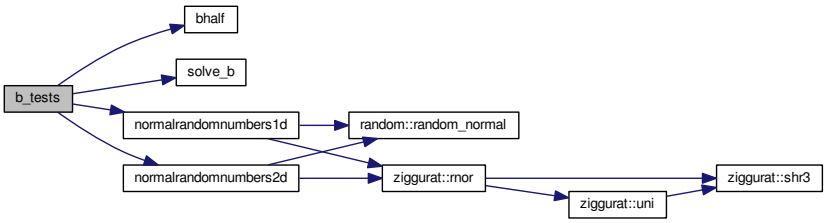
12.89.1 Function/Subroutine Documentation

12.89.1.1 subroutine b_tests ( )

These are some tests to check that the background error covariance matrix is implemented correctly

Definition at line 965 of file tests.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

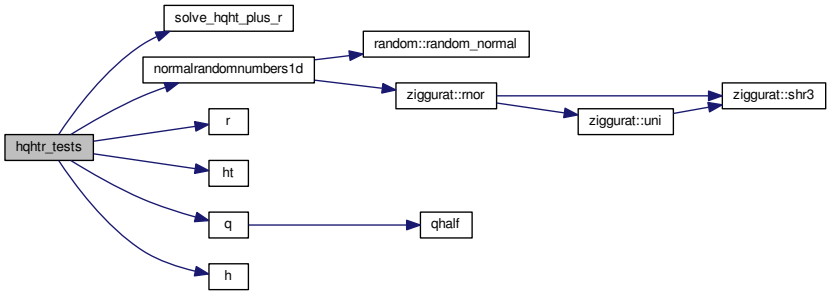
12.89.1.2 subroutine hqhtr_tests ( )

These are some tests to check that the linear solve operator is implemented correctly

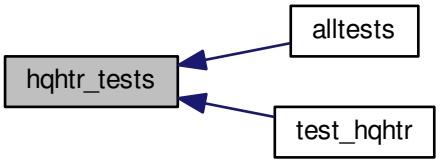
This should check the operation $(HQH^T + R)^{-1}$ is working

Definition at line 881 of file tests.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

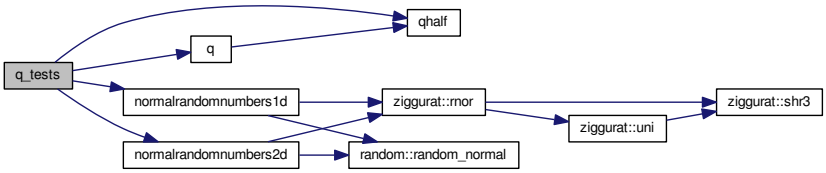


12.89.1.3 subroutine q_tests ( )

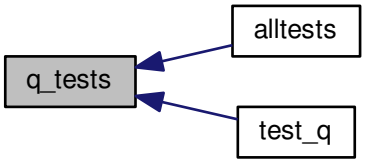
These are some tests to check that the model error covariance matrix is implemented correctly

Definition at line 675 of file tests.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

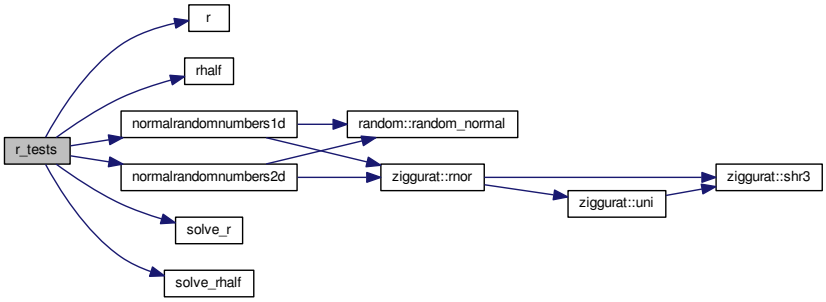


12.89.1.4 subroutine r_tests ( )

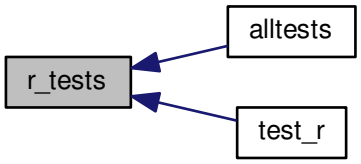
These are some tests to check that the observation error covariance matrix is implemented correctly

Definition at line 257 of file tests.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.90 src/user/model/model_as_subroutine_data.f90 File Reference

Data Types

- module model_as_subroutine_data

  a module that can be used to store the data for when the model is a subroutine of empire, i.e. using comms_version 4

12.91 src/user/model/model_as_subroutine_initialise.f90 File Reference

Functions/Subroutines

- subroutine model_as_subroutine_initialise (x, particle)

  subroutine to initialise an ensemble member when the model is a suboutine of EMPIRE, i.e. when using comms_↩ version = 4

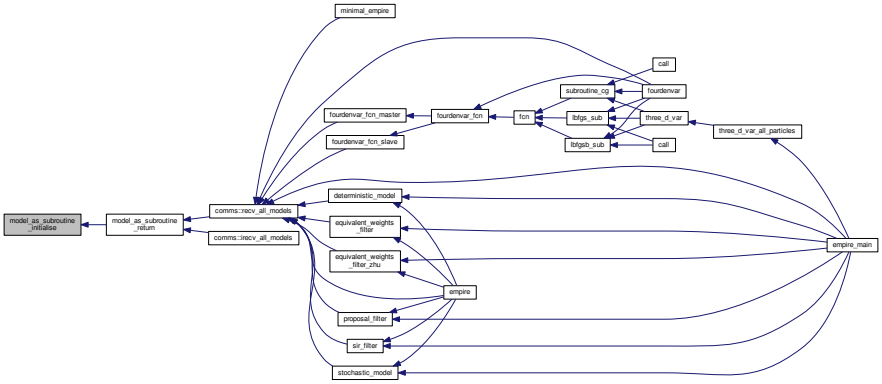
12.91.1 Function/Subroutine Documentation

12.91.1.1 subroutine model_as_subroutine_initialise (real(kind=kind(1.0d0)), dimension(state_dim), intent(out) *x*, integer, intent(in) *particle*)

subroutine to initialise an ensemble member when the model is a suboutine of EMPIRE, i.e. when using comms_↩ version = 4

Definition at line 31 of file model_as_subroutine_initialise.f90.

Here is the caller graph for this function:



12.92 src/user/model/model_as_subroutine_return.f90 File Reference

Functions/Subroutines

- subroutine model_as_subroutine_return (x, particle)

  subroutine to initialise and return the state from the model

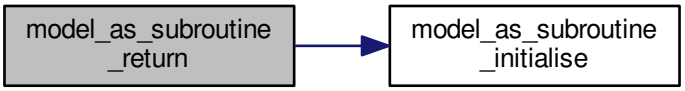
12.92.1 Function/Subroutine Documentation

12.92.1.1 subroutine model_as_subroutine_return (real(kind=kind(1.0d0)), dimension(state_dim), intent(out) *x*, integer, intent(in) *particle*)

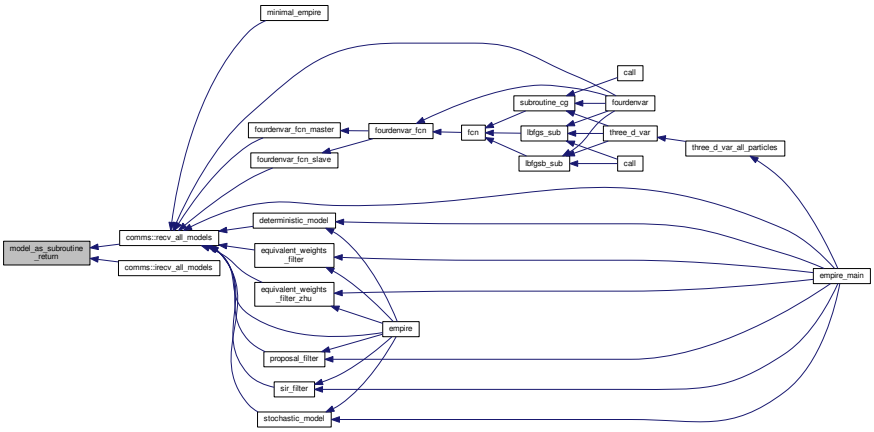
subroutine to initialise and return the state from the model

Definition at line 30 of file model_as_subroutine_return.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.93 src/user/model/model_as_subroutine_start.f90 File Reference

Functions/Subroutines

- subroutine model_as_subroutine_start (x, particle, tag)

  subroutine to increment the model when the model is a subroutine of empire. This is comms_v4 routine.

12.93.1 Function/Subroutine Documentation

12.93.1.1 subroutine model_as_subroutine_start (real(kind=kind(1.0d0)), dimension(state_dim), intent(in) *x,* integer, intent(in) *particle,* integer, intent(in) *tag*)

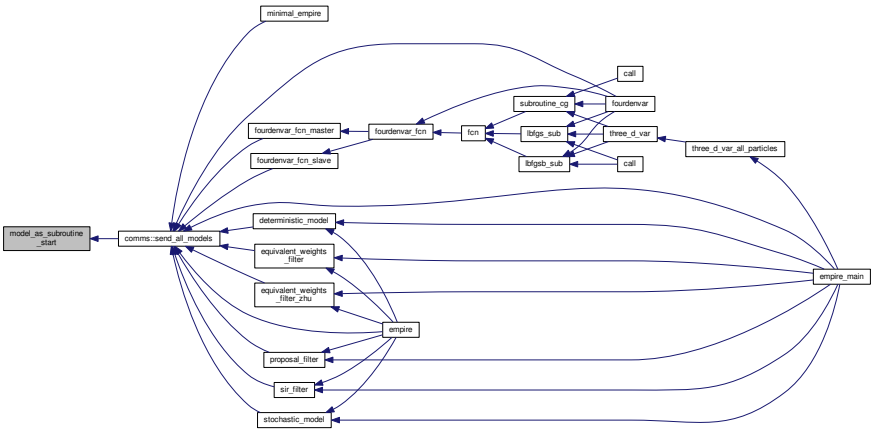
subroutine to increment the model when the model is a subroutine of empire. This is comms_v4 routine.

Parameters

out	*model_↩ states(,particle)*	
in	*x*	the input state vector to be incremented
in	*particle*	the particle number
in	*tag*	the tag that controls whether to continue, finish or reset time

Definition at line 32 of file model_as_subroutine_start.f90.

Here is the caller graph for this function:



12.94 src/user/Qdata.f90 File Reference

Data Types

- module qdata

  Module as a place to store user specified data for Q.

12.95 src/user/Rdata.f90 File Reference

Data Types

- module rdata

  Module to hold user supplied data for R observation error covariance matrix.
- module hqht_plus_r

12.96 src/user/user_initialise_mpi.f90 File Reference

Functions/Subroutines

- subroutine user_initialise_mpi

  Subroutine to initialise mpi in a special way if the model is weird like HadCM3 for example.
- subroutine user_mpi_send (stateDim, nrhs, x, tag)
- subroutine user_mpi_recv (stateDim, nrhs, x)
- subroutine user_mpi_irecv (stateDim, nrhs, x, requests)

12.96.1 Function/Subroutine Documentation

12.96.1.1 subroutine user_initialise_mpi ( )

Subroutine to initialise mpi in a special way if the model is weird like HadCM3 for example.

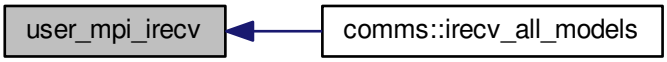
Definition at line 3 of file user_initialise_mpi.f90.

Here is the caller graph for this function:

12.96.1.2 subroutine user_mpi_irecv (integer, intent(in) *stateDim,* integer, intent(in) *nrhs,* real(kind=kind(1.0d0)), dimension(statedim,nrhs), intent(out) *x,* integer, dimension(nrhs), intent(inout) *requests*)

Definition at line 26 of file user_initialise_mpi.f90.

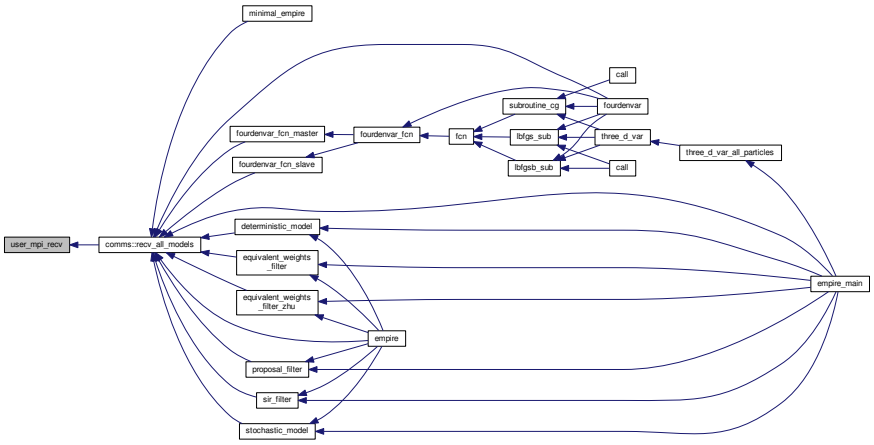
Here is the caller graph for this function:



12.96.1.3 subroutine user_mpi_recv (integer, intent(in) *stateDim,* integer, intent(in) *nrhs,* real(kind=kind(1.0d0)), dimension(statedim,nrhs), intent(out) *x*)

Definition at line 19 of file user_initialise_mpi.f90.

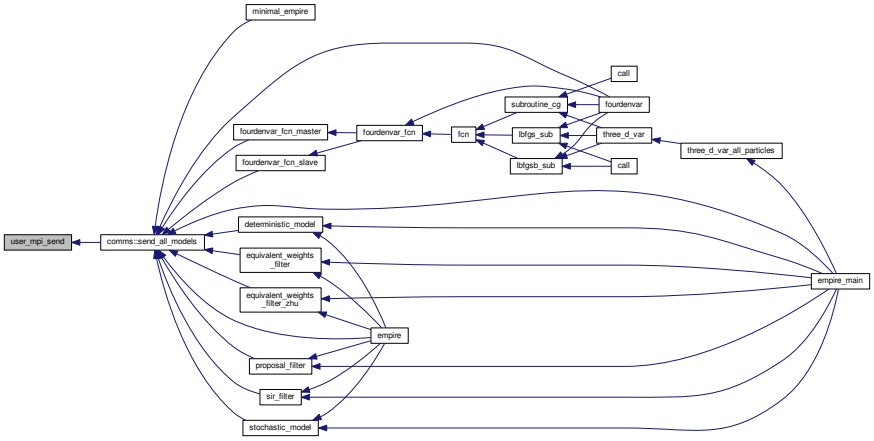
Here is the caller graph for this function:



12.96.1.4 subroutine user_mpi_send (integer, intent(in) *stateDim,* integer, intent(in) *nrhs,* real(kind=kind(1.0d0)), dimension(statedim,nrhs), intent(in) *x,* integer, intent(in) *tag*)

Definition at line 11 of file user_initialise_mpi.f90.

Here is the caller graph for this function:



12.97 src/user/user_perturb_particle.f90 File Reference

Functions/Subroutines

- subroutine user_perturb_particle (n, x)

  Subroutine to perturb state vector as defined by the user governed by the init option.

12.97.1 Function/Subroutine Documentation

12.97.1.1 subroutine user_perturb_particle (integer, intent(in) *n,* real(kind=rk), dimension(n), intent(inout) *x*)

Subroutine to perturb state vector as defined by the user governed by the init option.

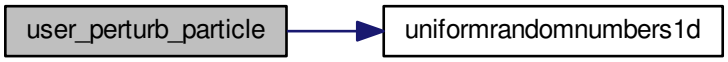
This should be considered an example routine. Here I shall implement a perturbation with a uniform variable on the interval $[-10, 15]$

Parameters

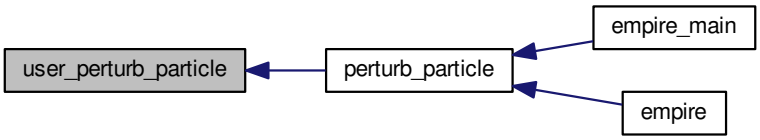
in	*n*	the dimension of the state vector x
in,out	*x*	the state to be perturbed

Definition at line 34 of file user_perturb_particle.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.98 src/utils/allocate_pf.f90 File Reference

Functions/Subroutines

- subroutine allocate_pf

  subroutine to allocate space for the filtering code

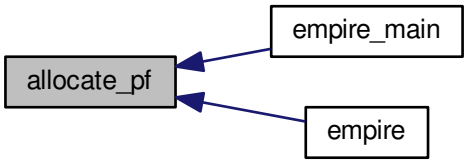
12.98.1 Function/Subroutine Documentation

12.98.1.1 subroutine allocate_pf ( )

subroutine to allocate space for the filtering code

Definition at line 28 of file allocate_pf.f90.

Here is the caller graph for this function:



12.99 src/utils/comms.f90 File Reference

Data Types

- module comms

  Module containing EMPIRE coupling data.

12.100 src/utils/data_io.f90 File Reference

Functions/Subroutines

- subroutine default_get_observation_data (y, t)

  Subroutine to read observation from a file
  Uses pftimestep to determine which observation to read.

- subroutine save_observation_data (y)

  Subroutine to save observation to a file
  Uses pftimestep to determine which observation to save.

- subroutine get_truth (x)

  Subroutine to read truth from the file written by save_truth
  .

- subroutine save_truth (x)

  Subroutine to save truth to a file
  .

- subroutine output_from_pf

  subroutine to output data from the filter

- subroutine save_state (state, filename)

  subroutine to save the state vector to a named file as an unformatted fortran file

- subroutine get_state (state, filename)

  subroutine to read the state vector from a named file as an unformatted fortran file

12.100.1 Function/Subroutine Documentation

12.100.1.1 subroutine default_get_observation_data (real(kind=rk), dimension(obs_dim), intent(out) *y*, integer, intent(in) *t*)

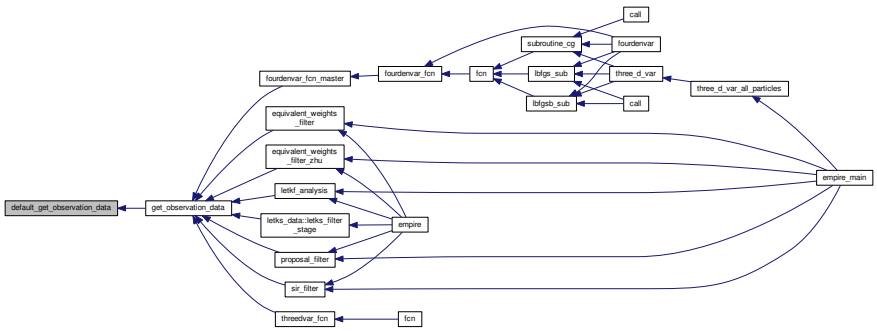
Subroutine to read observation from a file
Uses pftimestep to determine which observation to read.

Parameters

out	*y*	The observation
in	*t*	the current timestep

Definition at line 33 of file data_io.f90.

Here is the caller graph for this function:



12.100.1.2 subroutine get_state (real(kind=rk), dimension(state_dim), intent(out) *state,* character(256), intent(in) *filename*)

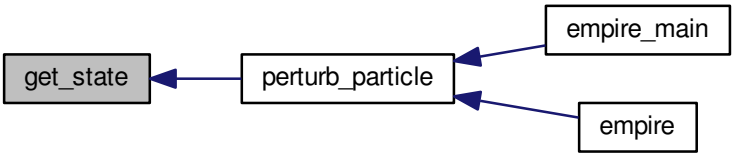
subroutine to read the state vector from a named file as an unformatted fortran file

Parameters

out	*state*	the state vector
in	*filename*	the name of the file to write the state vector in

Definition at line 258 of file data_io.f90.

Here is the caller graph for this function:



12.100.1.3 subroutine get_truth (real(kind=rk), dimension(state_dim), intent(out) *x*)

Subroutine to read truth from the file written by save_truth

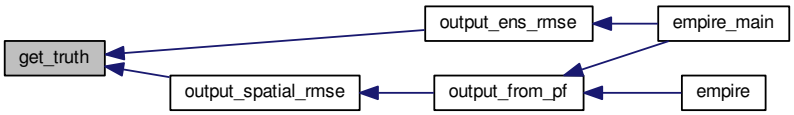
.

Parameters

out	*x*	The state vector

Definition at line 87 of file data_io.f90.

Here is the caller graph for this function:

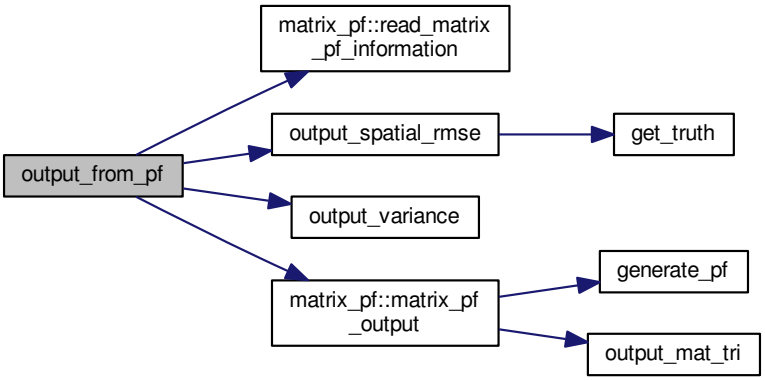


12.100.1.4 subroutine output_from_pf ( )

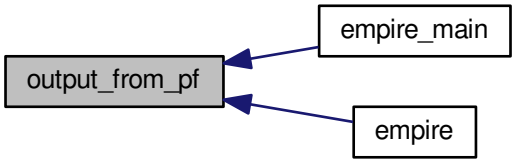
subroutine to output data from the filter

Definition at line 147 of file data_io.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.100.1.5 subroutine save_observation_data (real(kind=rk), dimension(obs_dim), intent(in) *y*)

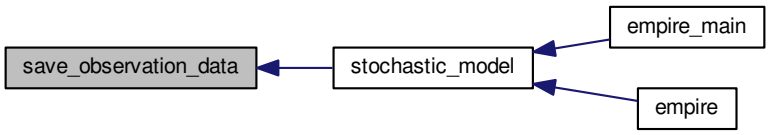
Subroutine to save observation to a file
Uses pftimestep to determine which observation to save.

Parameters

in	*y*	The observation

Definition at line 61 of file data_io.f90.

Here is the caller graph for this function:



12.100.1.6 subroutine save_state (real(kind=rk), dimension(state_dim), intent(in) *state,* character(256), intent(in) *filename*)

subroutine to save the state vector to a named file as an unformatted fortran file

Parameters

in	*state*	the state vector
in	*filename*	the name of the file to save the state vector in

Definition at line 231 of file data_io.f90.

12.100.1.7 subroutine save_truth (real(kind=rk), dimension(state_dim), intent(in) *x*)

Subroutine to save truth to a file

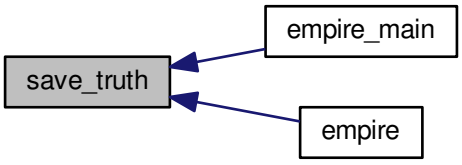
.

Parameters

in	*x*	The state vector

Definition at line 117 of file data_io.f90.

Here is the caller graph for this function:



12.101 src/utils/diagnostics.f90 File Reference

Functions/Subroutines

- subroutine diagnostics

  Subroutine to give output diagnositics such as rank histograms.

12.101.1 Function/Subroutine Documentation

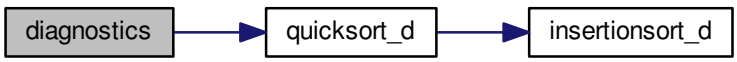
12.101.1.1 subroutine diagnostics ( )

Subroutine to give output diagnositics such as rank histograms.

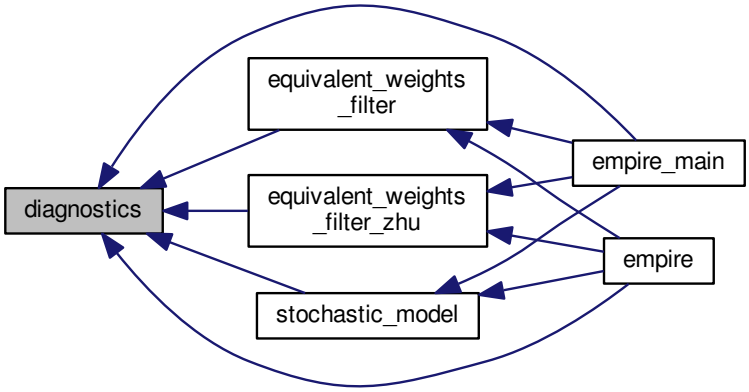
Todo test in anger with empire version 3. will probably segfault

Definition at line 31 of file diagnostics.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.102 src/utils/generate_pf.f90 File Reference

Functions/Subroutines

- subroutine generate_pf (stateDim, cnt, comm, x, pf)

  subroutine to generate Pf matrix given ensemble members on a communicator

12.102.1 Function/Subroutine Documentation

12.102.1.1 subroutine generate_pf (integer, intent(in) *stateDim,* integer, intent(in) *cnt,* integer, intent(in) *comm,* real(kind=rk), dimension(statedim,cnt), intent(in) *x,* real(kind=rk), dimension(statedim∗(statedim+1)/2), intent(out) *pf*)

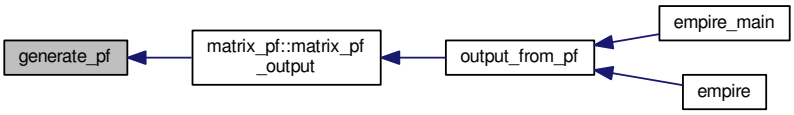
subroutine to generate Pf matrix given ensemble members on a communicator

Parameters

in	*statedim*	size of the state vectors
in	*cnt*	number of ensemble members on this process
in	*comm*	mpi communicator to use
in	*x*	the ensemble members on this process
out	*pf*	the Pf matrix i.e. the upper triangular part of the ensemble covariance matrix stored rectangular full packed form (see http://www.netlib.org/lapack/explore-html/db/d37/dtfttp_8f.html)

Definition at line 31 of file generate_pf.f90.

Here is the caller graph for this function:



12.103 src/utils/genQ.f90 File Reference

Functions/Subroutines

- subroutine genq

  Subroutine to estimate Q from a long model run.

12.103.1 Function/Subroutine Documentation

12.103.1.1 subroutine genq ( )

Subroutine to estimate Q from a long model run.

Definition at line 28 of file genQ.f90.

Here is the caller graph for this function:

12.104 src/utils/histogram.f90 File Reference

Data Types

- module histogram_data

  Module to control what variables are used to generate rank histograms.

12.105 src/utils/lambertw.f90 File Reference

Functions/Subroutines

- subroutine lambertw (K, X, W)

  subroutine to implement the lambertw function see https://en.wikipedia.org/wiki/Lambert_W_function

12.105.1 Function/Subroutine Documentation

12.105.1.1 subroutine lambertw (integer, intent(in) *K,* real(kind=kind(1.0d0)), intent(in) *X,* real(kind=kind(1.0d0)), intent(out) *W*)

subroutine to implement the lambertw function see https://en.wikipedia.org/wiki/Lambert_W_function

Definition at line 30 of file lambertw.f90.

Here is the caller graph for this function:

12.106 src/utils/loc_function.f90 File Reference

Functions/Subroutines

- subroutine loc_function (loctype, dis, scal, inc)

  subroutine to compute a localisation weighting based on a distance

12.106.1 Function/Subroutine Documentation

12.106.1.1 subroutine loc_function (integer, intent(in) *loctype,* real(kind=rk), intent(in) *dis,* real(kind=rk), intent(out) *scal,* logical, intent(out) *inc*)

subroutine to compute a localisation weighting based on a distance

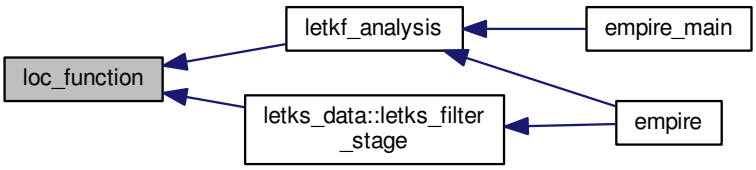
Todo include multiple localisation functions such as Gaspari-Cohn ones

Parameters

in	*loctype*	the choice of localisation function
in	*dis*	the input distance
out	*scal*	the localisation weighting to use
out	*inc*	logical signifying whether to compute with this sized weight or not.

Definition at line 31 of file loc_function.f90.

Here is the caller graph for this function:



12.107 src/utils/matrix_pf.f90 File Reference

Data Types

- module matrix_pf

  module to deal with generating and outputting pf matrix

- type matrix_pf::matrix_pf_data

12.108 src/utils/output_ens_rmse.f90 File Reference

Functions/Subroutines

- subroutine output_ens_rmse ()

  subroutine to output RMSEs

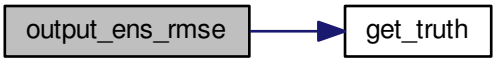
12.108.1 Function/Subroutine Documentation

12.108.1.1 subroutine output_ens_rmse ( )

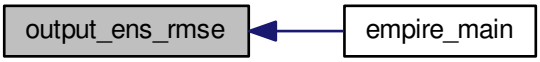
subroutine to output RMSEs

Definition at line 29 of file output_ens_rmse.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.109 src/utils/output_mat_tri.f90 File Reference

Functions/Subroutines

- subroutine output_mat_tri (n, A, filename, output_type)

  subroutine to output triangluar matrix various formats

12.109.1 Function/Subroutine Documentation

12.109.1.1 subroutine output_mat_tri (integer, intent(in) *n,* real(kind=rk), dimension(n∗(n+1)/2), intent(in) *A,* character(40), intent(in) *filename,* integer, intent(in) *output_type*)

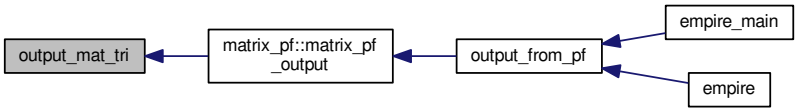
subroutine to output triangluar matrix various formats

Parameters

in	*n*	number of columns of matrix A
in	*a*	matrix to be output in rectangular full packed format (TF)
in	*filename*	the name of the file to be output
in	*output_type*	output file type. • 0 - undefined • 1 - standard packed format (TP) • 2 - rectangular full packed format (TF) Negative values will be formatted. Positive values will be unformatted.

Definition at line 29 of file output_mat_tri.f90.

Here is the caller graph for this function:



12.110 src/utils/output_spatial_rmse.f90 File Reference

Functions/Subroutines

- subroutine output_spatial_rmse (mean)

  subroutine to output RMSEs

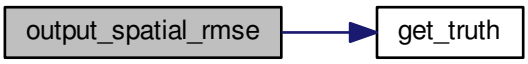
12.110.1 Function/Subroutine Documentation

12.110.1.1 subroutine output_spatial_rmse (real(kind=rk), dimension(state_dim), intent(in) *mean*)

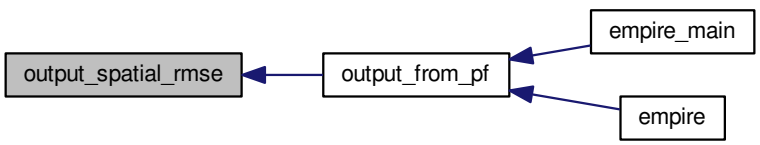
subroutine to output RMSEs

Definition at line 29 of file output_spatial_rmse.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.111 src/utils/output_variance.f90 File Reference

Functions/Subroutines

- subroutine output_variance (mean)

  subroutine to output ensemble variance

12.111.1 Function/Subroutine Documentation

12.111.1.1 subroutine output_variance (real(kind=rk), dimension(state_dim), intent(in) *mean*)

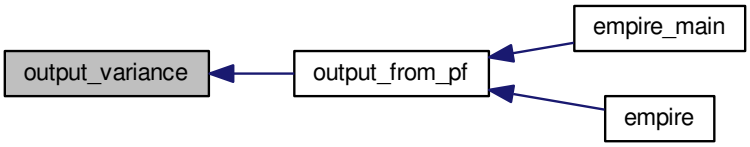
subroutine to output ensemble variance

Parameters

in	*mean*	the

Definition at line 29 of file output_variance.f90.

Here is the caller graph for this function:



12.112 src/utils/quicksort.f90 File Reference

Functions/Subroutines

- recursive subroutine quicksort_d (a, na)

  subroutine to sort using the quicksort algorithm

- subroutine insertionsort_d (A, nA)

  subroutine to sort using the insertionsort algorithm

12.112.1 Function/Subroutine Documentation

12.112.1.1 subroutine insertionsort_d (real(kind=kind(1.0d0)), dimension(na), intent(inout) *A*, integer, intent(in) *nA*)

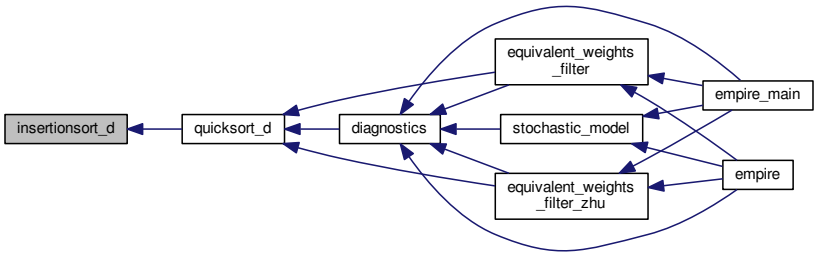
subroutine to sort using the insertionsort algorithm

Parameters

in,out	*a*	array of doubles to be sorted
in	*na*	dimension of array a

Definition at line 86 of file quicksort.f90.

Here is the caller graph for this function:



12.112.1.2 recursive subroutine quicksort_d (real(kind=kind(1.0d0)), dimension(na), intent(inout) *a,* integer, intent(in) *na*)

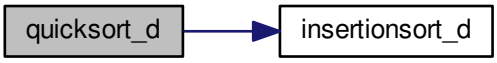
subroutine to sort using the quicksort algorithm

Parameters

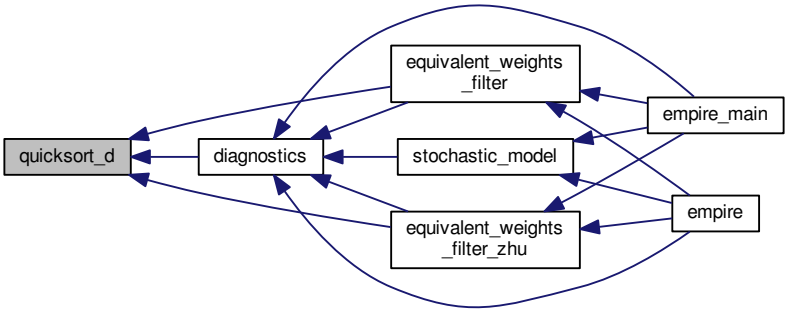
in,out	*a*	array of doubles to be sorted
in	*na*	dimension of array a

Definition at line 9 of file quicksort.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.113 src/utils/random_d.f90 File Reference

Data Types

- module random

  A module for random number generation from the following distributions:

12.114 src/utils/randperm.f90 File Reference

Functions/Subroutines

- subroutine randperm (N, p)

  subroutine to create an array of a random permutations of the natural numbers from 1 to N

12.114.1 Function/Subroutine Documentation

12.114.1.1 subroutine randperm (integer, intent(in) *N*, integer, dimension(n), intent(out) *p*)

subroutine to create an array of a random permutations of the natural numbers from 1 to N

Parameters

in	*n*	length of array P
out	*p*	on output, this is a random permutation of the integers from 1 to N

Definition at line 30 of file randperm.f90.

12.115 src/utils/trajectories.f90 File Reference

Data Types

- module traj_data

  module to hold data for trajectories

Functions/Subroutines

- subroutine trajectories

  subroutine to output trajectories

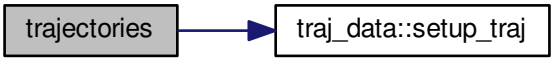
12.115.1 Function/Subroutine Documentation

12.115.1.1 subroutine trajectories ( )

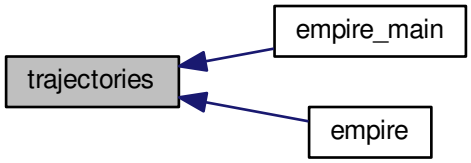
subroutine to output trajectories

Definition at line 145 of file trajectories.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.116 src/utils/ziggurat.f90 File Reference

Data Types

- module ziggurat

12.117 src/var/three_d_var.f90 File Reference

Functions/Subroutines

- subroutine three_d_var (x)

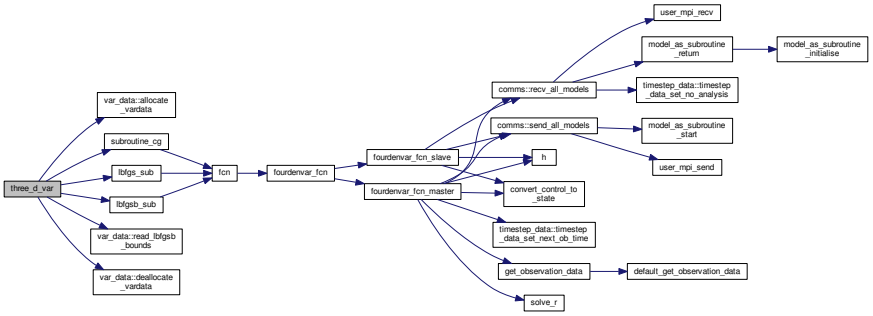
12.117.1 Function/Subroutine Documentation

12.117.1.1 subroutine three_d_var (real(kind=rk), dimension(state_dim), intent(inout) *x*)

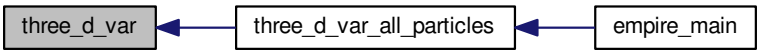
Todo make work with empire version 3

Definition at line 29 of file three_d_var.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



12.118 src/var/three_d_var_all_particles.f90 File Reference

Functions/Subroutines

- subroutine three_d_var_all_particles

  subroutine to call 3DVar for each particle

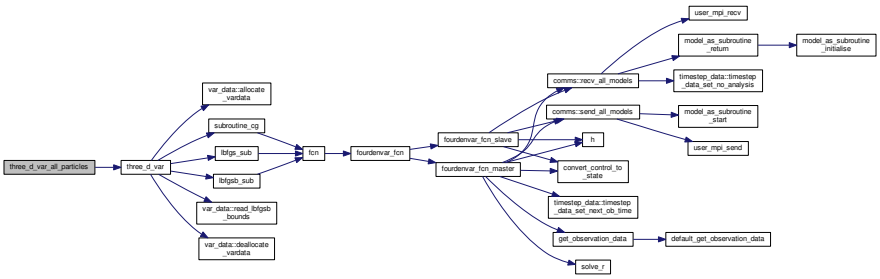
12.118.1 Function/Subroutine Documentation

12.118.1.1 subroutine three_d_var_all_particles ( )

subroutine to call 3DVar for each particle

Definition at line 2 of file three_d_var_all_particles.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

12.119 src/var/threedvar_data.f90 File Reference

Data Types

- module threedvar_data

  module to store stuff for 3DVar

12.120 src/var/threedvar_fcn.f90 File Reference

Functions/Subroutines

- subroutine threedvar_fcn (n, x, f, g)

  subroutine to provide the objective function and gradient for 3dvar

12.120.1 Function/Subroutine Documentation

12.120.1.1 subroutine threedvar_fcn (integer, intent(in) *n*, real(kind=rk), dimension(n), intent(in) *x*, real(kind=rk), intent(out) *f*, real(kind=rk), dimension(n), intent(out) *g*)

subroutine to provide the objective function and gradient for 3dvar

Let x be the state we wish to find using Var.

The objective function considered is

$J(x) = \frac{1}{2}(x-x_b)^T B^{-1}(x-x_b) + \frac{1}{2}(y-H(x))^T R^{-1}(y-H(x)))$

where x_b is a background guess, B the background error covariance matrix,

y are the observations, and H the corresponding observation operator with associated observation error covariance matrix R.

The gradient of the objective function can then be written

$g = \nabla J(x) \approx B^{-1}(x-x_b) - H^T R^{-1}(y-H(x))$

which is exact if H is linear

NOTE: this will only currently work for EMPIRE VERSION 1 of 2.

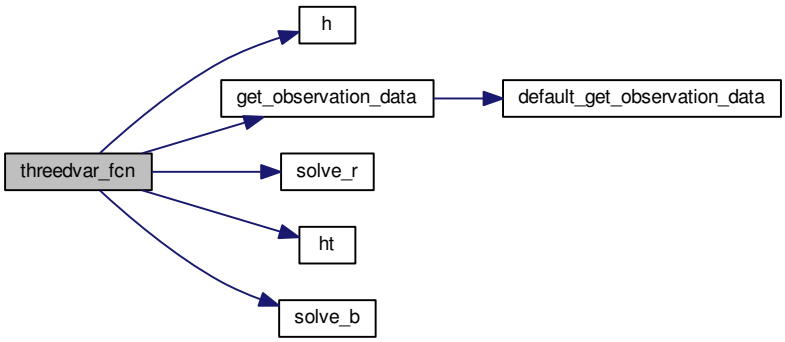
Todo update 3dvar to work with EMPIRE VERSION 3!

Parameters

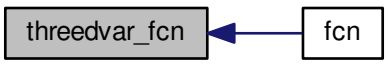
in	*n*	the dimension of the state
in	*x*	current guess
out	*g*	gradient of objective function
out	*f*	the objective function

Definition at line 54 of file threedvar_fcn.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



Index