EMPIRE DA

0.1

Generated by Doxygen 1.8.6

Tue Sep 23 2014 12:07:47

Contents

Chapter 1

EMPIRE Data Assimilation Documentation

Author

Philip A. Browne

Date

Time-stamp: <2014-09-23 12:07:45 pbrowne>

1.1 Downloading

These codes are hosted on www.bitbucket.org and can be attained with the following commands:

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

or

```
wget https://bitbucket.org/pbrowne/empire-data-assimilation/get/aa31fdfc3912.zip && gunzip aa31fdfc3912.zip
```

1.2 Compiling

1.2.1 Compilation of the source code

The Makefile must be editted for the specific compiler setup. In the main directory you will find the file `Makefile`. Edit the variables as follows:

- `FC` The fortran compiler
- `FCOPTS` The options for the fortran compiler
- `LIB_LIST` The libraries to be called. Note this must include BLAS

To compile the source code, simply then type the command

```
make
```

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

- empire

- alltests
- test_h
- test_hqhtr
- test_q
- test_r

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

```
make clean
```

1.2.2 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.

```
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

```
make doc_html
```

1.3 Customising for specific models

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

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.
- 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, $x = R^{-1}b$
- 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 $(HQH^T + R)$

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.

1.4 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 $HH^T x = x$, and if $Q^{\frac{1}{2}} Q^{\frac{1}{2}} x = Qx$ for various different vectors x.

1.5 Linking to your model using EMPIRE

Todo Write a bit about how to put the MPI commands into the model, or point to where that is. www.met.reading.ac.uk/~darc/empire

1.6 Running

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

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

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

Todo I have to talk about how pf_parameters.dat works.

1.7 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

Todo List

page EMPIRE Data Assimilation Documentation

Write a bit about how to put the MPI commands into the model, or point to where that is. `www.met.reading.ac.uk/~darc/empire`

I have to talk about how pf_parameters.dat works.

Chapter 3

Data Type Index

3.1 Data Types List

Here are the data types with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Type Documentation

5.1 comms Module Reference

Module containing EMPIRE coupling data.

Public Member Functions

- subroutine allocate_data
- subroutine deallocate_data
- subroutine initialise_mpi

  subroutine to make EMPIRE connections and saves details into pf_control module

Public Attributes

- integer cpl_mpi_comm
- integer mype_id
- integer myrank
- integer nproc
- integer pf_mpi_comm
- integer pfrank
- integer npfs
- integer, dimension(:), allocatable gblcount
- integer, dimension(:), allocatable gbldisp

5.1.1 Detailed Description

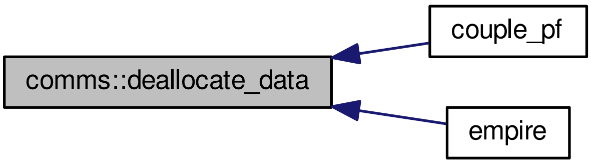
Module containing EMPIRE coupling data.

5.1.2 Member Function/Subroutine Documentation

5.1.2.1 subroutine comms::allocate_data ( )

5.1.2.2 subroutine comms::deallocate_data ( )

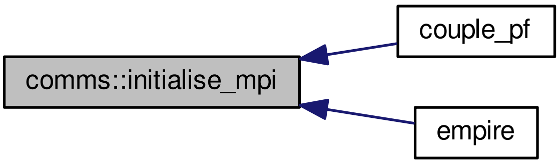
Here is the caller graph for this function:



5.1.2.3 subroutine comms::initialise_mpi ( )

subroutine to make EMPIRE connections and saves details into pf_control module

Here is the caller graph for this function:



5.1.3 Member Data Documentation

5.1.3.1 integer comms::cpl_mpi_comm

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

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

5.1.3.4 integer comms::mype_id

5.1.3.5 integer comms::myrank

5.1.3.6 integer comms::npfs

5.1.3.7 integer comms::nproc

5.1.3.8 integer comms::pf_mpi_comm

5.1.3.9 integer comms::pfrank

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

- src/utils/comms.f90

5.2 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 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

5.2.1 Detailed Description

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

5.2.2 Member Function/Subroutine Documentation

5.2.2.1 subroutine histogram_data::kill_histogram_data ( )

subroutine to clean up arrays used in rank histograms

5.2.2.2 subroutine histogram_data::load_histogram_data ( )

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

5.2.3 Member Data Documentation

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

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

5.2.3.3 integer histogram_data::rhl_n

5.2.3.4 integer histogram_data::rhn_n

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

- src/utils/histogram.f90

5.3 hqht_plus_r Module Reference

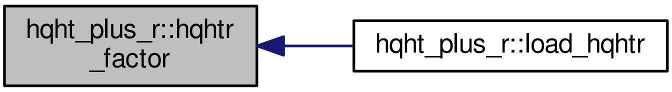
Public Member Functions

- subroutine load_hqhtr
- subroutine hqhtr_factor
- subroutine kill_hqhtr

5.3.1 Member Function/Subroutine Documentation

5.3.1.1 subroutine hqht_plus_r::hqhtr_factor ( )

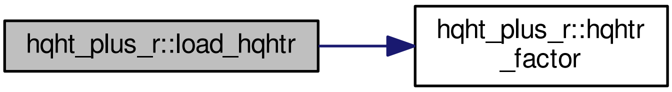
Here is the caller graph for this function:



5.3.1.2 subroutine hqht_plus_r::kill_hqhtr ( )

5.3.1.3 subroutine hqht_plus_r::load_hqhtr ( )

Here is the call graph for this function:



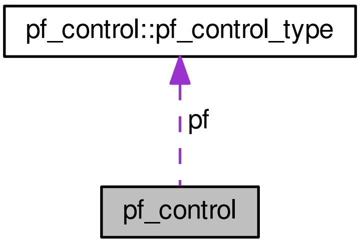
The documentation for this module was generated from the following file:

- src/data/Rdata.f90

5.4 pf_control Module Reference

module to hold 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 allocate_pf
- subroutine deallocate_pf

Public Attributes

- type(pf_control_type) pf

  the derived data type holding all controlling data

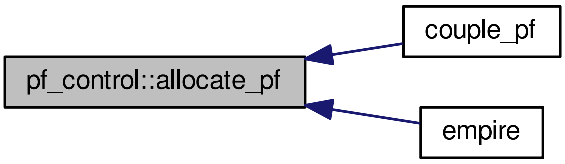
5.4.1 Detailed Description

module to hold all the information to control the the main program

5.4.2 Member Function/Subroutine Documentation

5.4.2.1 subroutine pf_control::allocate_pf ( )

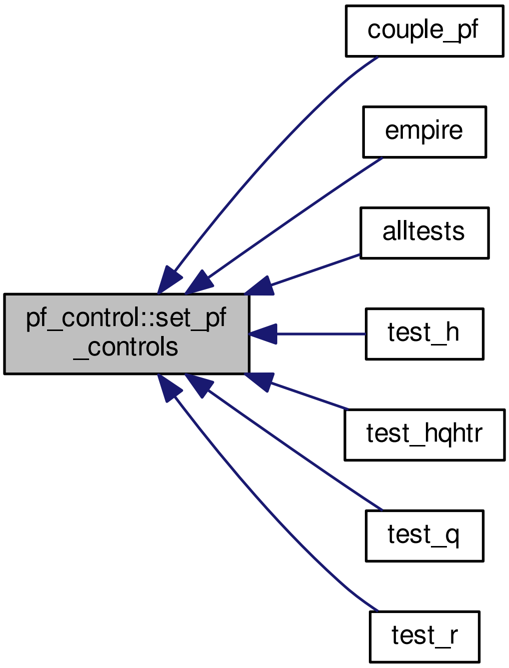
Here is the caller graph for this function:



5.4.2.2 subroutine pf_control::deallocate_pf ( )

5.4.2.3 subroutine pf_control::set_pf_controls ( )

Here is the caller graph for this function:



5.4.3 Member Data Documentation

5.4.3.1 type(pf_control_type) pf_control::pf

the derived data type holding all controlling data

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

- src/controlers/pf_control.f90

5.5 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
- logical human_readable

  unused
- 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
- integer couple_root

  empire master processor
- logical use_talagrand

  switch if true outputs rank histograms
- logical use_weak

  switch unused

- logical use_mean

  switch if true outputs ensemble mean
- logical use_var

  switch if true outputs ensemble variance
- logical use_traj

  switch if true outputs trajectories
- logical use_rmse

  switch if true outputs Root Mean Square Errors
- 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) type

  which filter to use
- character(1) init

  which method to initialise ensemble

5.5.1 Member Data Documentation

5.5.1.1 integer pf_control::pf_control_type::count

number of ensemble members associated with this MPI process

5.5.1.2 integer pf_control::pf_control_type::couple_root

empire master processor

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

5.5.1.4 logical pf_control::pf_control_type::gen_data

true generates synthetic obs for a twin experiment

5.5.1.5 logical pf_control::pf_control_type::gen_q

true attempts to build up Q from long model run

5.5.1.6 logical pf_control::pf_control_type::human_readable

unused

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

which method to initialise ensemble

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

proportion of particles to keep in EWPF EW step

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

mean state vector

5.5.1.10 integer pf_control::pf_control_type::nens

the total number of ensemble members

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

standard deviation of normal distribution in mixture density

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

the nudging factor

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

particles associates with this MPI process

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

state vector of ensemble members on this mpi process

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

scalar to multiply Q by

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

storage for rank histograms

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

dunno

5.5.1.18 integer pf_control::pf_control_type::time_bwn_obs

the number of model timesteps between observations

5.5.1.19 integer pf_control::pf_control_type::time_obs

the number of observations we will assimilate

5.5.1.20 integer pf_control::pf_control_type::timestep =0

the current timestep as the model progresses

5.5.1.21 character(2) pf_control::pf_control_type::type

which filter to use

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

half width of the uniform distribution in mixture density

5.5.1.23 logical pf_control::pf_control_type::use_mean

switch if true outputs ensemble mean

5.5.1.24 logical pf_control::pf_control_type::use_rmse

switch if true outputs Root Mean Square Errors

5.5.1.25 logical pf_control::pf_control_type::use_talagrand

switch if true outputs rank histograms

5.5.1.26 logical pf_control::pf_control_type::use_traj

switch if true outputs trajectories

5.5.1.27 logical pf_control::pf_control_type::use_var

switch if true outputs ensemble variance

5.5.1.28 logical pf_control::pf_control_type::use_weak

switch unused

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

the negative log of the weights of the particles

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

- src/controlers/pf_control.f90

5.6 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

Public Attributes

- integer qn
- integer qne
- integer, dimension(:), allocatable qrow
- integer, dimension(:), allocatable qcol
- real(kind=kind(1.0d0)), dimension(:), allocatable qval
- real(kind=kind(1.0d0)), dimension(:), allocatable qdiag
- real(kind=kind(1.0d0)) qscale

5.6.1 Detailed Description

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

- the model error covariance matrix

5.6.2 Member Function/Subroutine Documentation

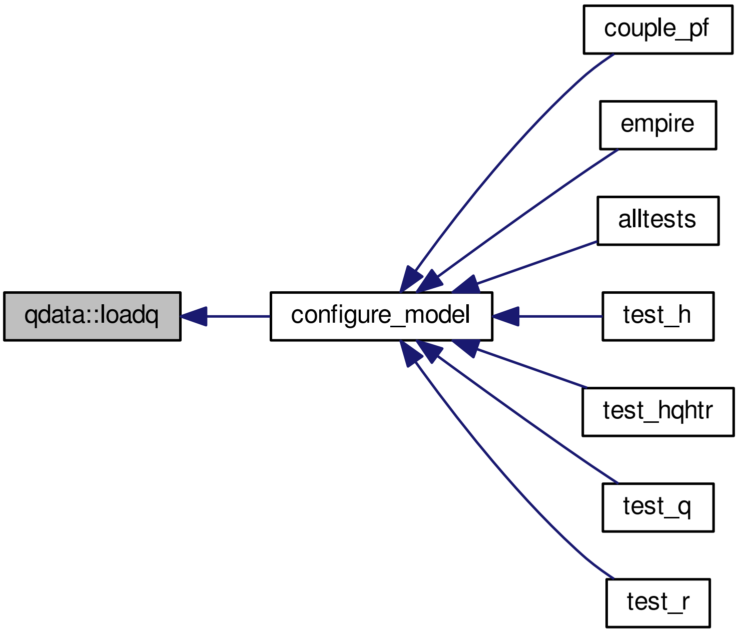
5.6.2.1 subroutine qdata::killq ( )

SUbroutine to deallocate user data for Q

5.6.2.2 subroutine qdata::loadq ( )

Subroutine to load in user data for Q.

Here is the caller graph for this function:



5.6.3 Member Data Documentation

5.6.3.1 integer, dimension(:), allocatable qdata::qcol

5.6.3.2 real(kind=kind(1.0d0)), dimension(:), allocatable qdata::qdiag

5.6.3.3 integer qdata::qn

5.6.3.4 integer qdata::qne

5.6.3.5 integer, dimension(:), allocatable qdata::qrow

5.6.3.6 real(kind=kind(1.0d0)) qdata::qscale

5.6.3.7 real(kind=kind(1.0d0)), dimension(:), allocatable qdata::qval

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

- src/data/Qdata.f90

5.7 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)

5.7.1 Detailed Description

A module for random number generation from the following distributions:

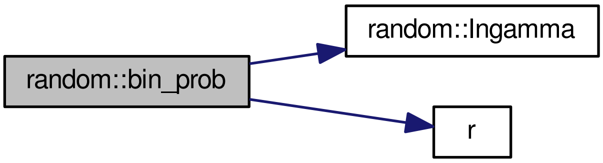
Distribution Function/subroutine name

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

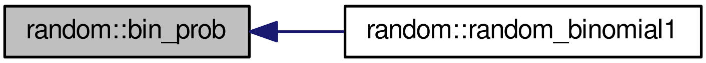
5.7.2 Member Function/Subroutine Documentation

5.7.2.1 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*)

Here is the call graph for this function:

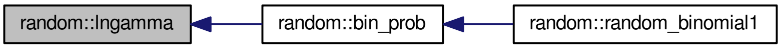


Here is the caller graph for this function:



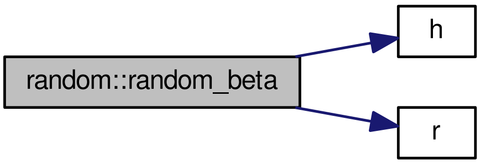
5.7.2.2 real (dp) function random::lngamma (real (dp), intent(in) *x*)

Here is the caller graph for this function:



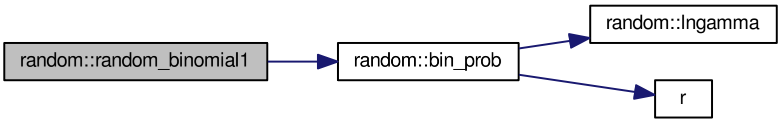
5.7.2.3 real(kind=kind(1.0d+0)) function random::random_beta (real(kind=kind(1.0d+0)), intent(in) *aa*, real(kind=kind(1.0d+0)), intent(in) *bb*, logical, intent(in) *first*)

Here is the call graph for this function:



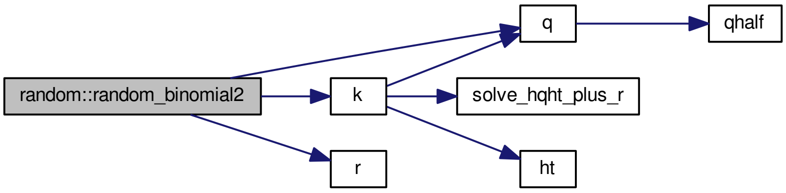
5.7.2.4 integer function random::random_binomial1 (integer, intent(in) *n*, real(kind=kind(1.0d+0)), intent(in) *p*, logical, intent(in) *first*)

Here is the call graph for this function:



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

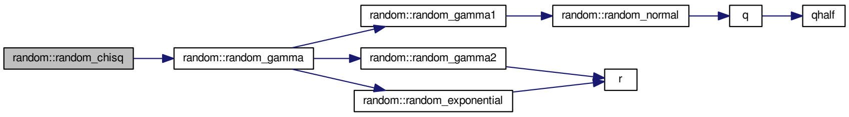
Here is the call graph for this function:



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

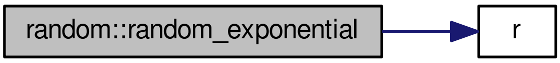
5.7.2.7 real(kind=kind(1.0d+0)) function random::random_chisq (integer, intent(in) *ndf,* logical, intent(in) *first*)

Here is the call graph for this function:

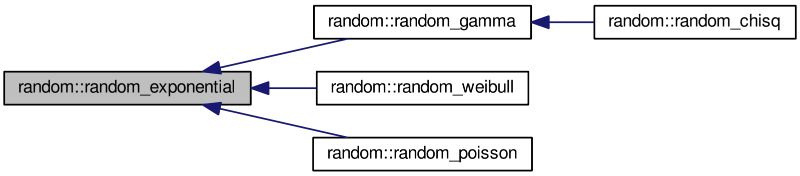


5.7.2.8 real(kind=kind(1.0d+0)) function random::random_exponential ( )

Here is the call graph for this function:

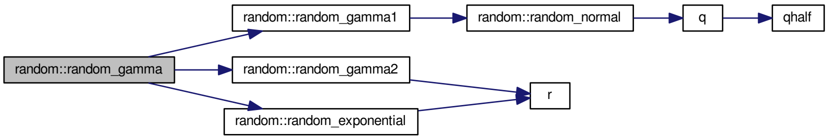


Here is the caller graph for this function:

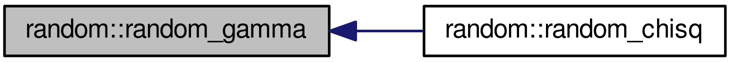


5.7.2.9 real(kind=kind(1.0d+0)) function random::random_gamma (real(kind=kind(1.0d+0)), intent(in) *s,* logical, intent(in) *first*)

Here is the call graph for this function:

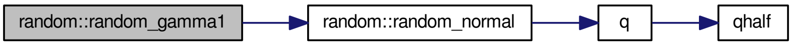


Here is the caller graph for this function:

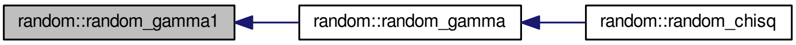


5.7.2.10 real(kind=kind(1.0d+0)) function random::random_gamma1 (real(kind=kind(1.0d+0)), intent(in) *s*, logical, intent(in) *first*)

Here is the call graph for this function:

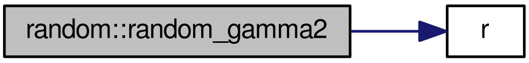


Here is the caller graph for this function:

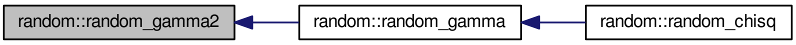


5.7.2.11 real(kind=kind(1.0d+0)) function random::random_gamma2 (real(kind=kind(1.0d+0)), intent(in) *s*, logical, intent(in) *first*)

Here is the call graph for this function:

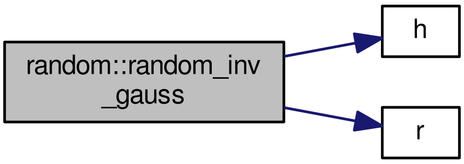


Here is the caller graph for this function:



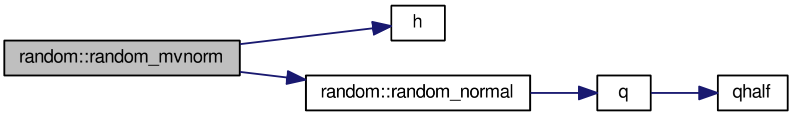
5.7.2.12 real(kind=kind(1.0d+0)) function random::random_inv_gauss (real(kind=kind(1.0d+0)), intent(in) *h*, real(kind=kind(1.0d+0)), intent(in) *b*, logical, intent(in) *first*)

Here is the call graph for this function:



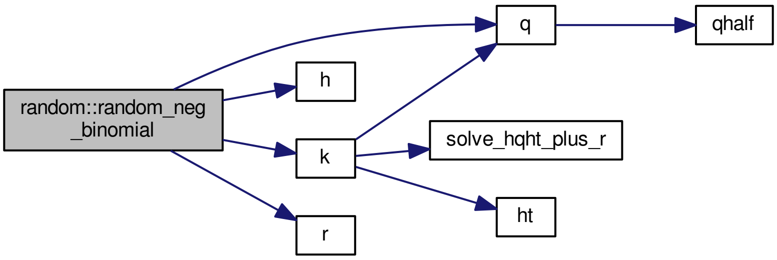
5.7.2.13 subroutine random::random_mvnorm (integer, intent(in) *n*, real(kind=kind(1.0d+0)), dimension(:), intent(in) *h*, real(kind=kind(1.0d+0)), dimension(:), intent(in) *d*, real(kind=kind(1.0d+0)), dimension(:), intent(inout) *f*, logical, intent(in) *first*, real(kind=kind(1.0d+0)), dimension(:),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

Here is the call graph for this function:



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

Here is the call graph for this function:



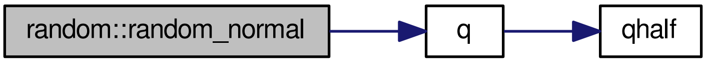
5.7.2.15 real(kind=kind(1.0d+0)) function random::random_normal ( )

function to get random normal with zero mean and stdev 1

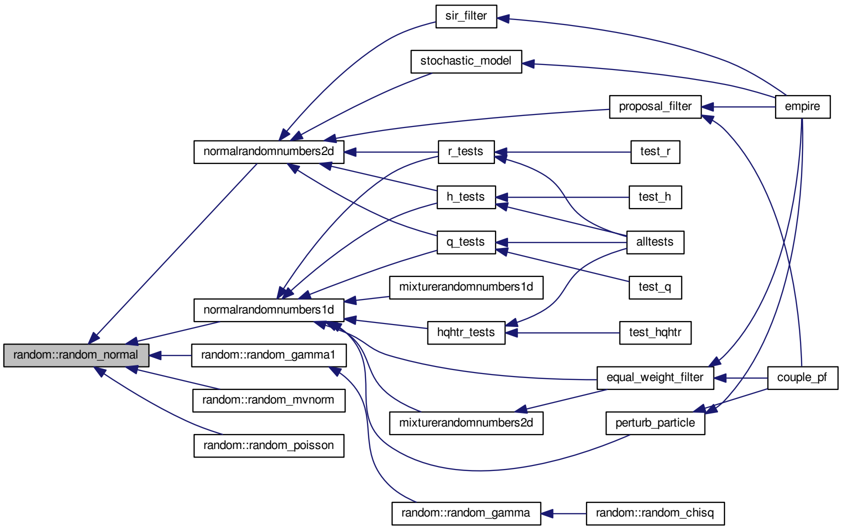
Returns

fn_val

Here is the call graph for this function:

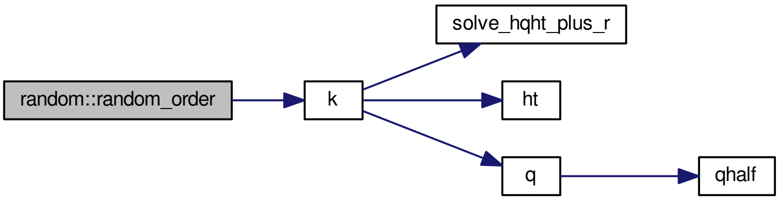


Here is the caller graph for this function:



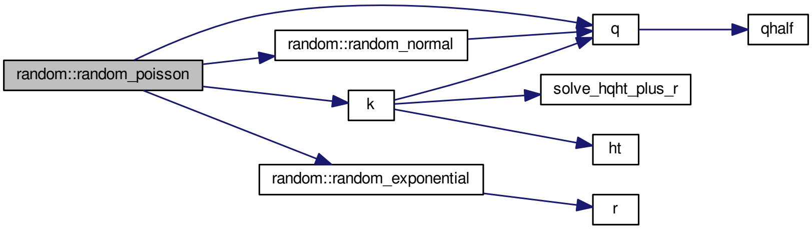
5.7.2.16 subroutine random::random_order (integer, dimension(n), intent(out) *order,* integer, intent(in) *n*)

Here is the call graph for this function:



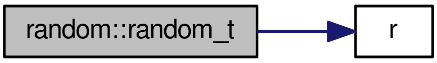
5.7.2.17 integer function random::random_poisson (real(kind=kind(1.0d+0)), intent(in) *mu,* logical, intent(in) *first*)

Here is the call graph for this function:



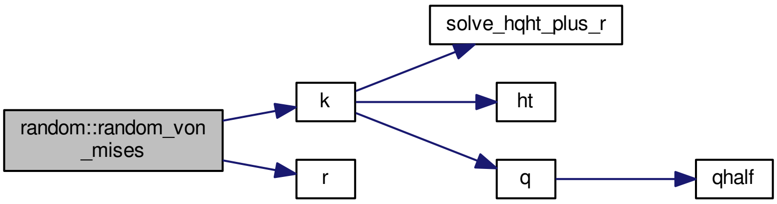
5.7.2.18 real(kind=kind(1.0d+0)) function random::random_t (integer, intent(in) *m*)

Here is the call graph for this function:



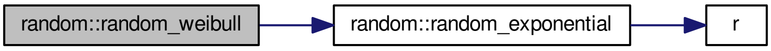
5.7.2.19 real(kind=kind(1.0d+0)) function random::random_von_mises (real(kind=kind(1.0d+0)), intent(in) *k,* logical, intent(in) *first*)

Here is the call graph for this function:



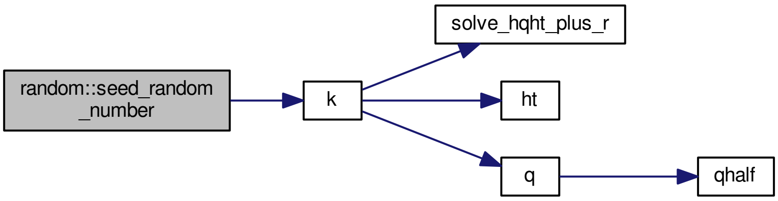
5.7.2.20 real(kind=kind(1.0d+0)) function random::random_weibull (real(kind=kind(1.0d+0)), intent(in) *a*)

Here is the call graph for this function:



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

Here is the call graph for this function:



5.7.3 Member Data Documentation

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

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

- src/utils/random_d.f90

5.8 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

Public Attributes

- integer rn
- integer rne

- integer, dimension(:), allocatable rrow
- integer, dimension(:), allocatable rcol
- real(kind=kind(1.0d0)), dimension(:), allocatable rval
- real(kind=kind(1.0d0)), dimension(:), allocatable rdiag

5.8.1 Detailed Description

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

5.8.2 Member Function/Subroutine Documentation

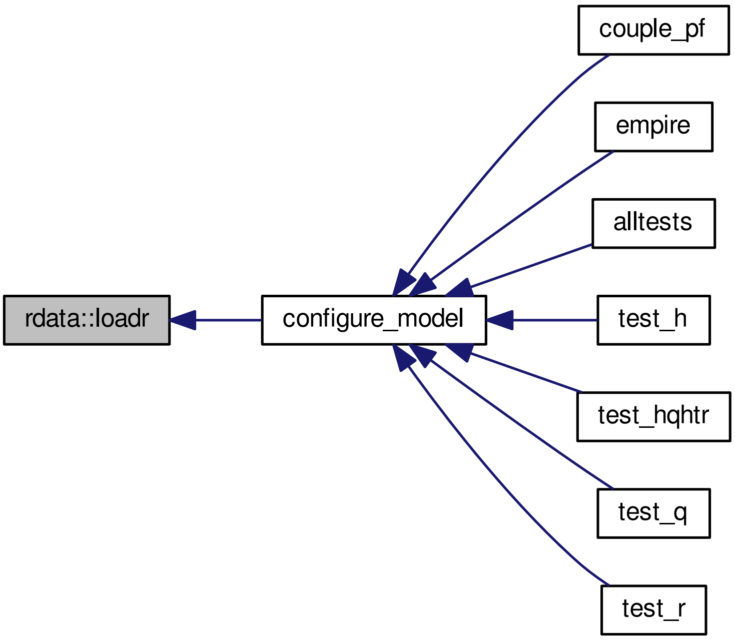
5.8.2.1 subroutine rdata::killr ( )

SUbroutine to deallocate R data

5.8.2.2 subroutine rdata::loadr ( )

Subroutine to load data for R.

Here is the caller graph for this function:



5.8.3 Member Data Documentation

5.8.3.1 integer, dimension(:), allocatable rdata::rcol

5.8.3.2 real(kind=kind(1.0d0)), dimension(:), allocatable rdata::rdiag

5.8.3.3 integer rdata::rn

5.8.3.4 integer rdata::rne

5.8.3.5 integer, dimension(:), allocatable rdata::rrow

5.8.3.6 real(kind=kind(1.0d0)), dimension(:), allocatable rdata::rval

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

- src/data/Rdata.f90

5.9 sizes Module Reference

Module that stores the dimension of observation and state spaces.

Public Attributes

- integer obs_dim

  size of the observation space
- integer state_dim

  dimension of the model

5.9.1 Detailed Description

Module that stores the dimension of observation and state spaces.

5.9.2 Member Data Documentation

5.9.2.1 integer sizes::obs_dim

size of the observation space

5.9.2.2 integer sizes::state_dim

dimension of the model

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

- src/controlers/sizes.f90

Chapter 6

File Documentation

6.1 model_specific.f90 File Reference

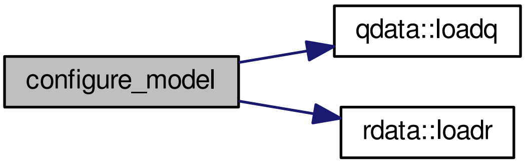
Functions/Subroutines

- subroutine configure_model
- subroutine solve_r (y, v, t)
- subroutine solve_hqht_plus_r (y, v, t)
- subroutine q (nrhs, x, Qx)
- subroutine qhalf (nrhs, x, Qx)
- subroutine r (nrhs, y, Ry, t)
- subroutine rhalf (nrhs, y, Ry, t)
- subroutine h (x, hx, t)
- subroutine ht (y, x, t)

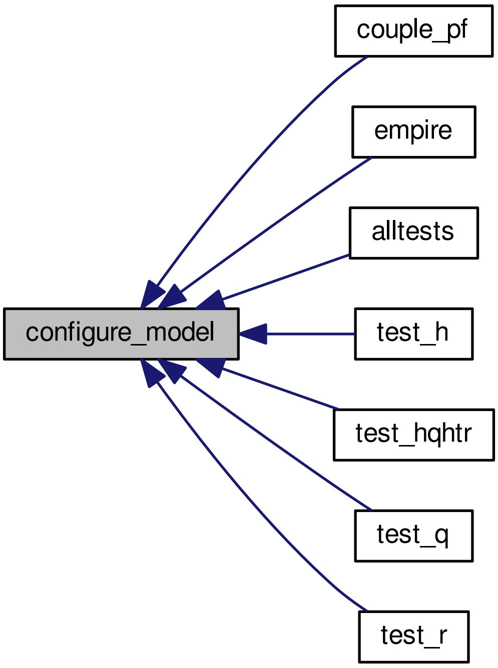
6.1.1 Function/Subroutine Documentation

6.1.1.1 subroutine configure_model ( )

Here is the call graph for this function:

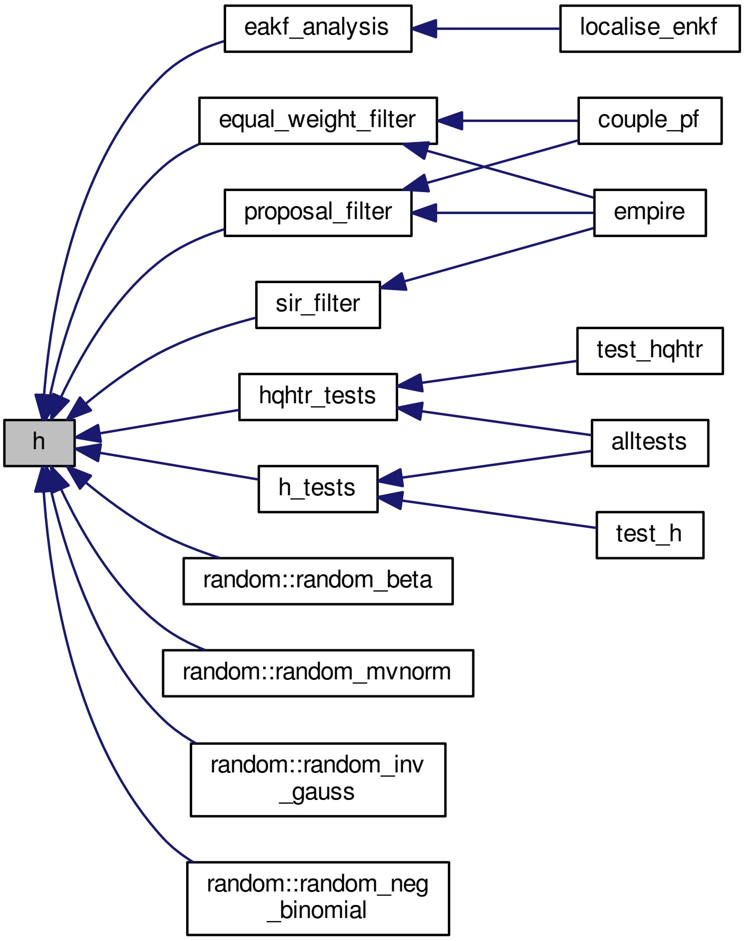


Here is the caller graph for this function:



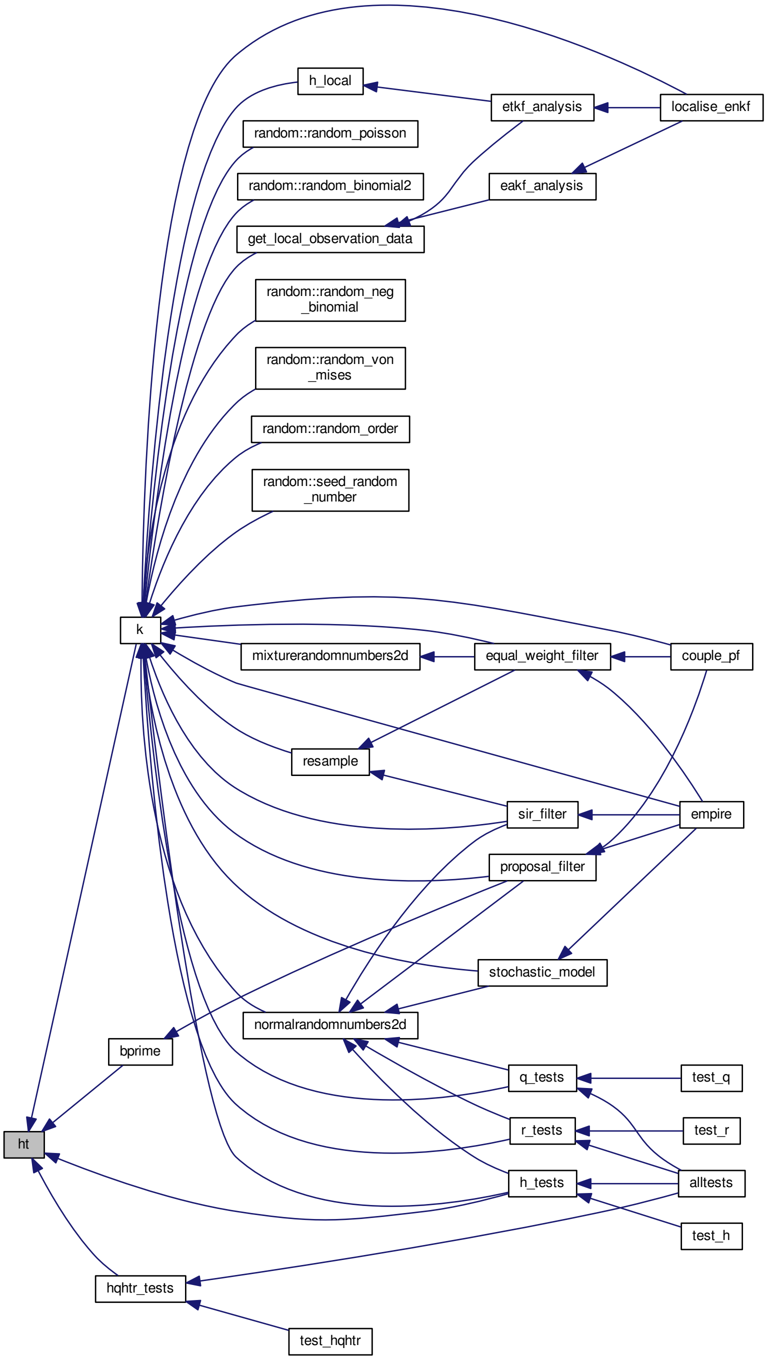
6.1.1.2 subroutine h (real(kind=rk), dimension(state_dim,pf%count), intent(in) *x*, real(kind=rk), dimension(obs_dim,pf%count), intent(out) *hx*, integer, intent(in) *t*)

Here is the caller graph for this function:



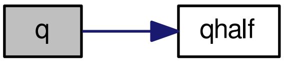
6.1.1.3 subroutine ht (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y*, real(kind=rk), dimension(state_dim,pf%count), intent(out) *x*, integer, intent(in) *t*)

Here is the caller graph for this function:

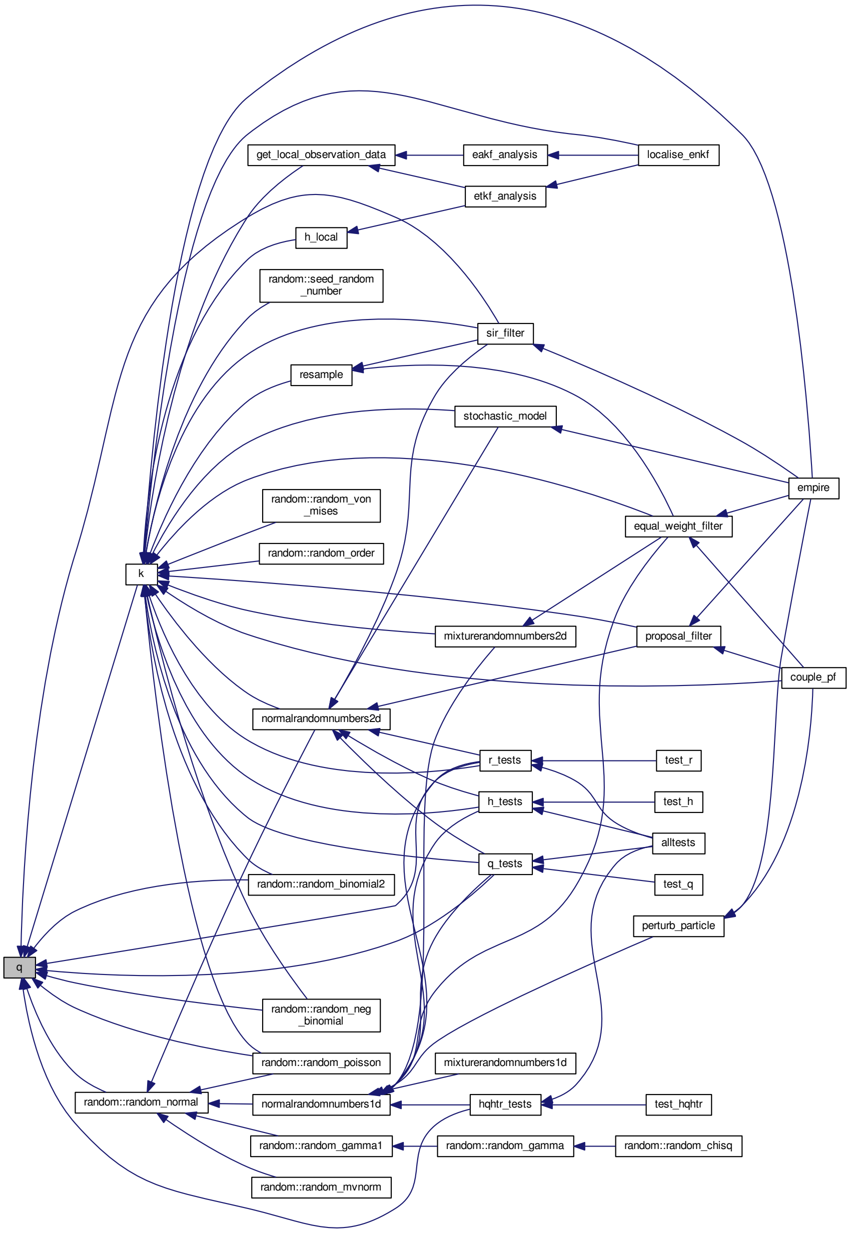


6.1.1.4 subroutine q (integer, intent(in) *nrhs*, real(kind=rk), dimension(state_dim,nrhs), intent(in) *x*, real(kind=rk), dimension(state_dim,nrhs), intent(out) *Qx*)

Here is the call graph for this function:

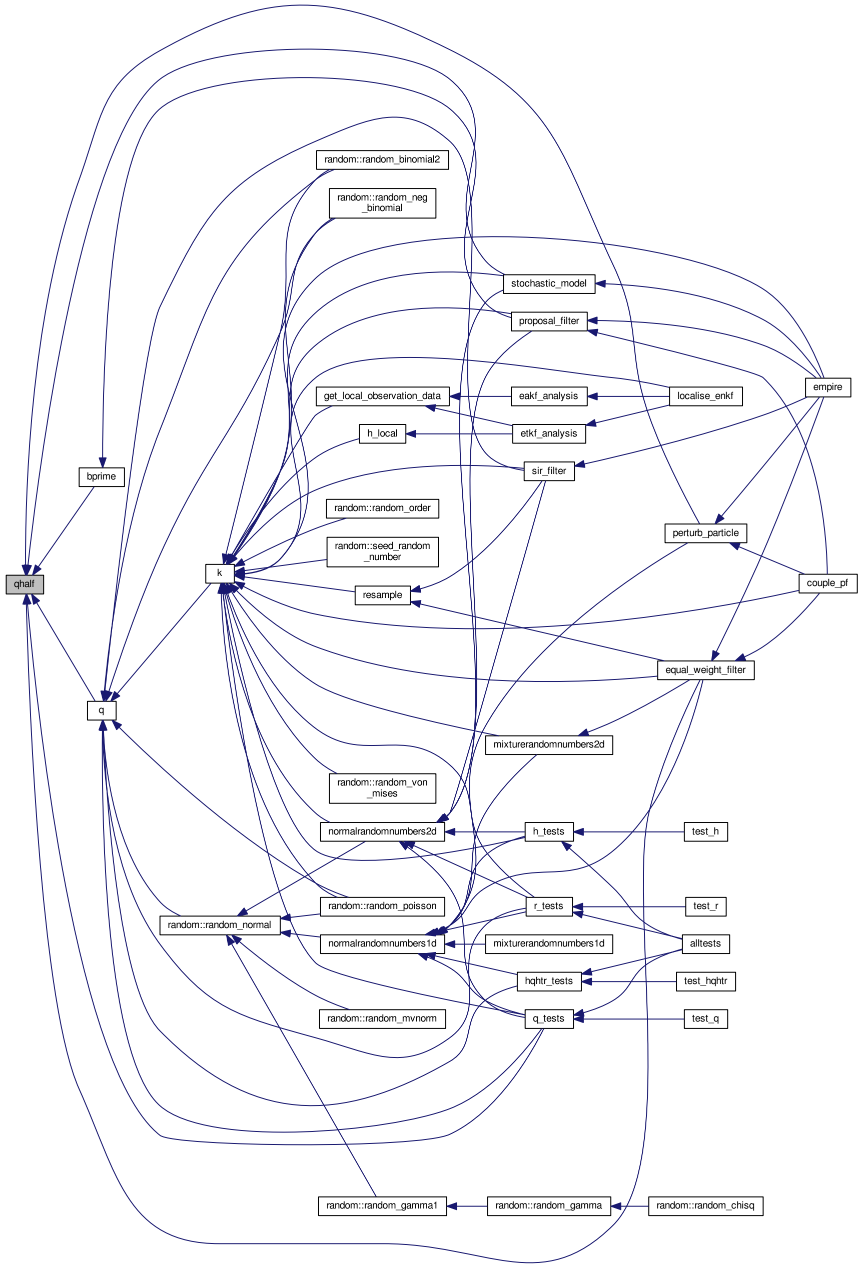


Here is the caller graph for this function:



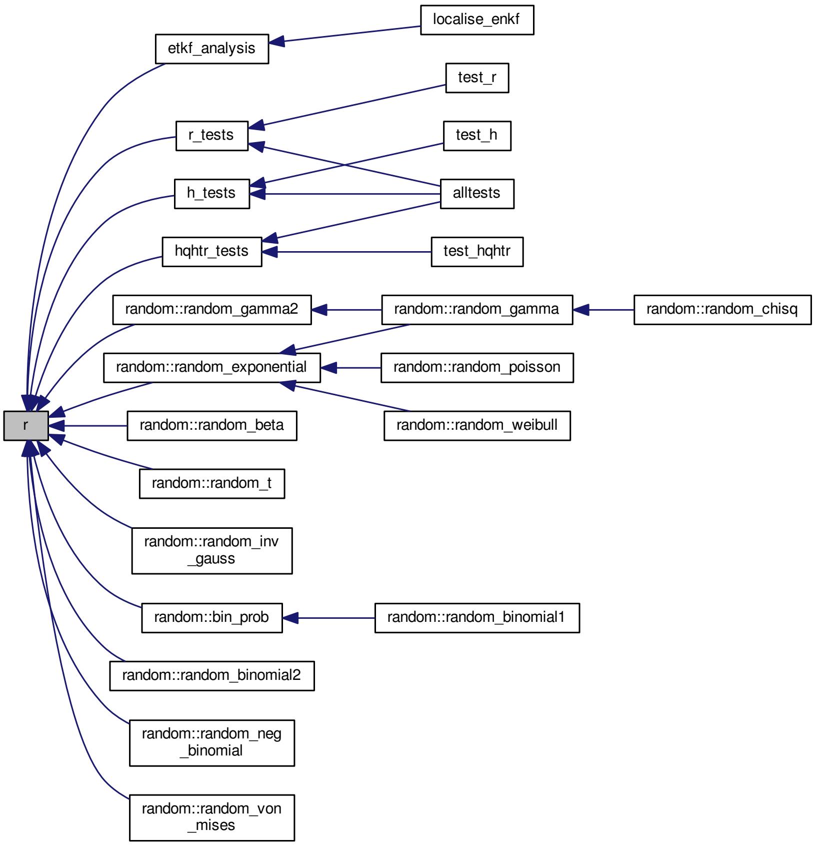
6.1.1.5 subroutine qhalf (integer, intent(in) *nrhs,* real(kind=rk), dimension(state_dim,nrhs), intent(in) *x,* real(kind=rk), dimension(state_dim,nrhs), intent(out) *Qx*)

Here is the caller graph for this function:



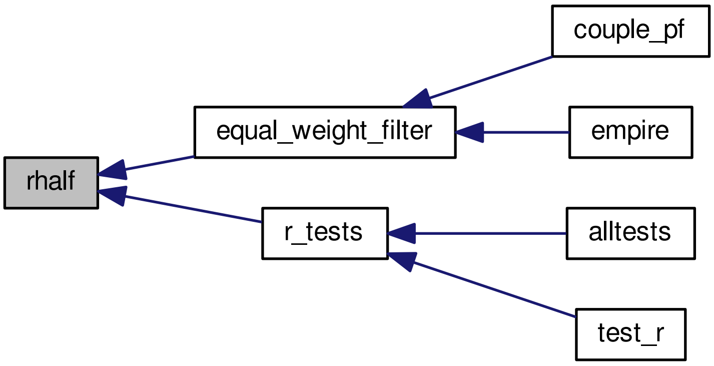
6.1.1.6 subroutine r (integer, intent(in) *nrhs,* real(kind=rk), dimension(obs_dim,nrhs), intent(in) *y,* real(kind=rk), dimension(obs_dim,nrhs), intent(out) *Ry,* integer, intent(in) *t*)

Here is the caller graph for this function:



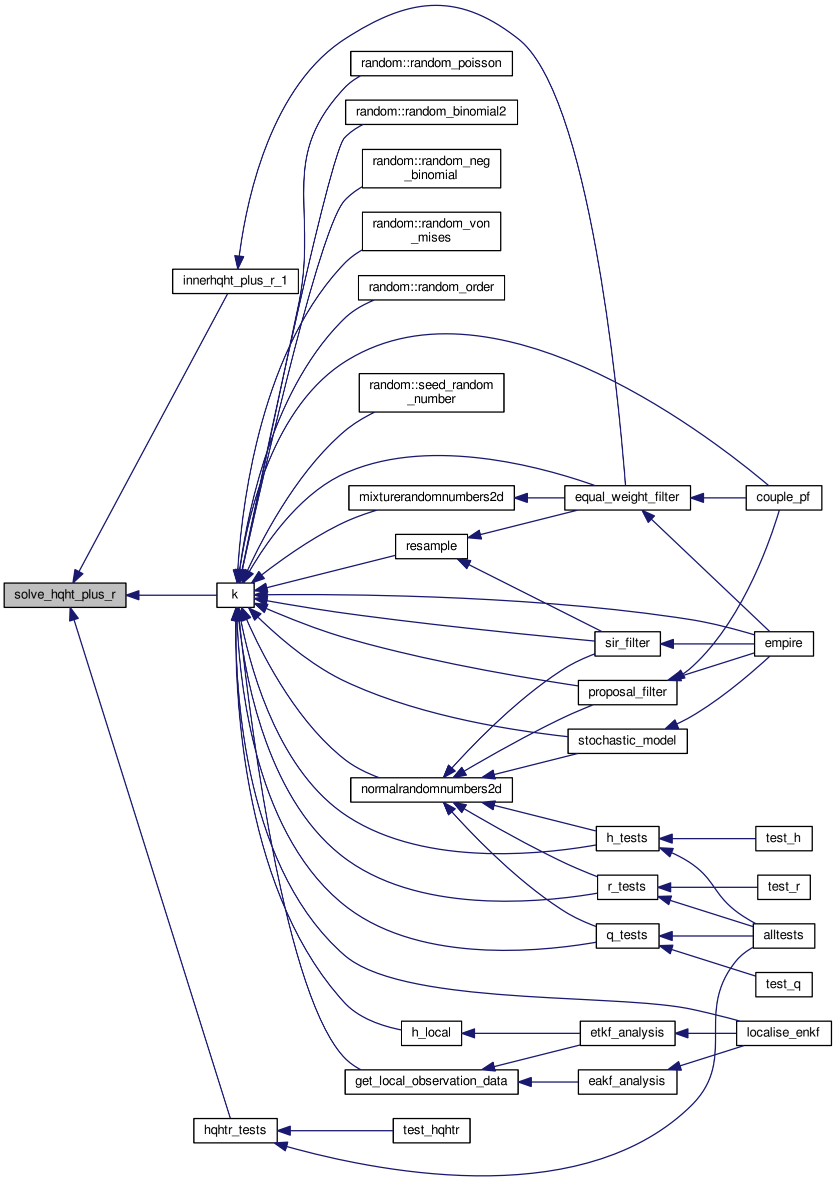
6.1.1.7 subroutine rhalf (integer, intent(in) *nrhs,* real(kind=rk), dimension(obs_dim,nrhs), intent(in) *y,* real(kind=rk), dimension(obs_dim,nrhs), intent(out) *Ry,* integer, intent(in) *t*)

Here is the caller graph for this function:



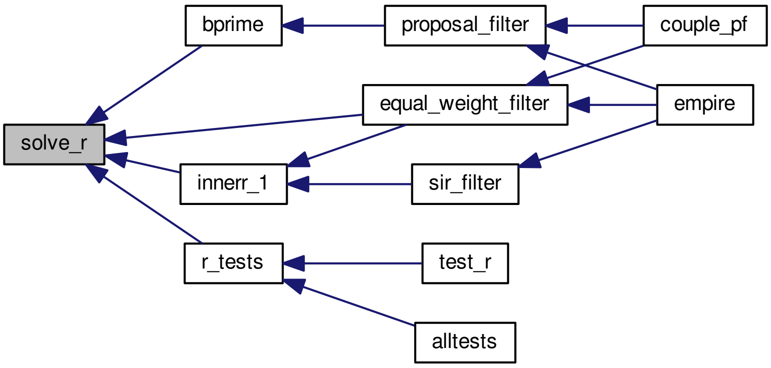
6.1.1.8 subroutine solve_hqht_plus_r (real(kind=rk), dimension(obs_dim), intent(in) *y*, real(kind=rk), dimension(obs_dim), intent(out) *v*, integer, intent(in) *t*)

Here is the caller graph for this function:



6.1.1.9 subroutine solve_r (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y*, real(kind=rk), dimension(obs_dim,pf%count), intent(out) *v*, integer, intent(in) *t*)

Here is the caller graph for this function:



6.2 src/controlers/old_pf_couple.f90 File Reference

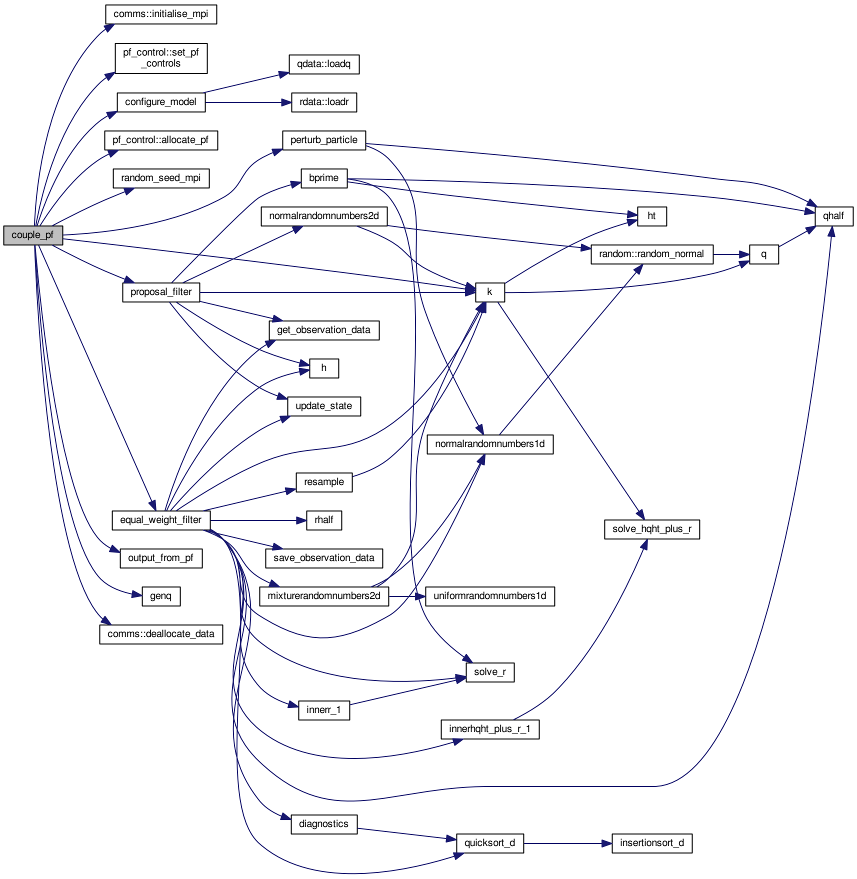
Functions/Subroutines

- program couple_pf

6.2.1 Function/Subroutine Documentation

6.2.1.1 program couple_pf ( )

Here is the call graph for this function:



6.3 src/controlers/pf_control.f90 File Reference

Data Types

- module pf_control

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

6.4 src/controlers/pf_couple.f90 File Reference

Functions/Subroutines

- program empire

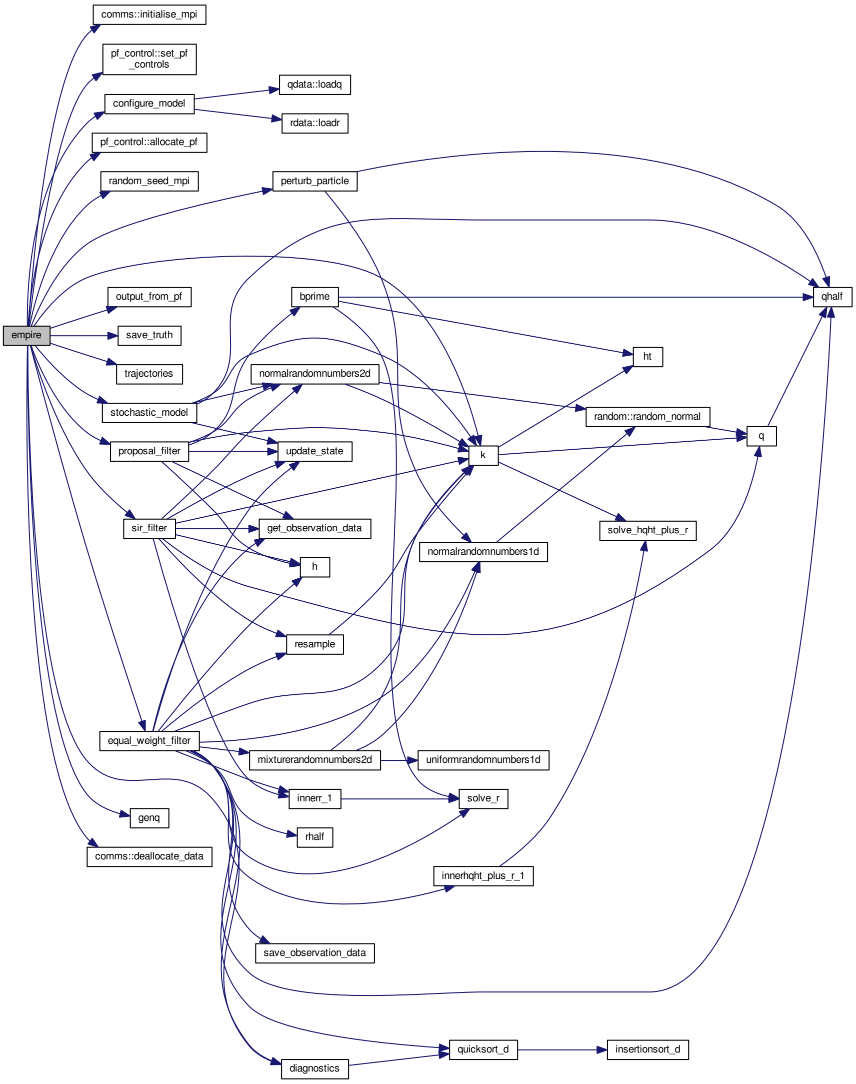
  the main program

6.4.1 Function/Subroutine Documentation

6.4.1.1 program empire ( )

the main program

Here is the call graph for this function:



6.5 src/controlers/sizes.f90 File Reference

Data Types

- module sizes

    Module that stores the dimension of observation and state spaces.

6.6 src/data/Qdata.f90 File Reference

Data Types

- module qdata

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

6.7 src/data/Rdata.f90 File Reference

Data Types

- module rdata

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

6.8 src/DOC_README.txt File Reference

6.9 src/filters/eakf_analysis.f90 File Reference

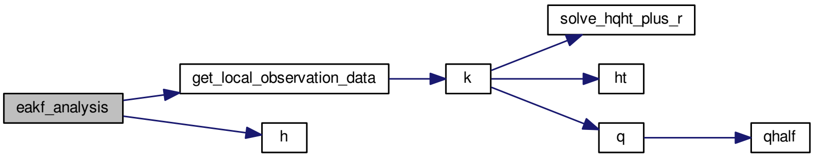
Functions/Subroutines

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

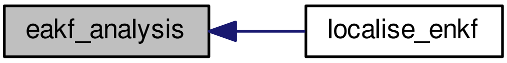
6.9.1 Function/Subroutine Documentation

6.9.1.1 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*)

Here is the call graph for this function:



Here is the caller graph for this function:



6.10 src/filters/enkf_specific.f90 File Reference

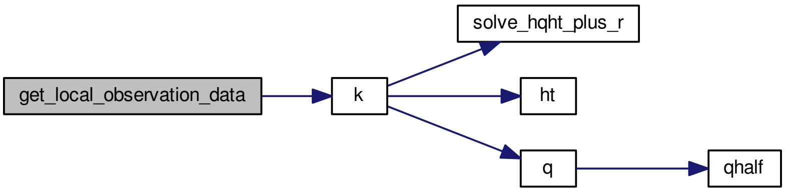
Functions/Subroutines

- 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)

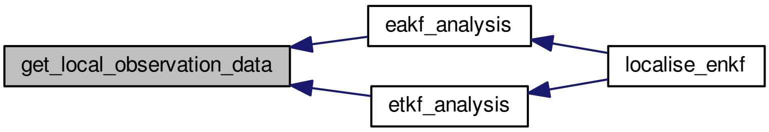
6.10.1 Function/Subroutine Documentation

6.10.1.1 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*)

Here is the call graph for this function:

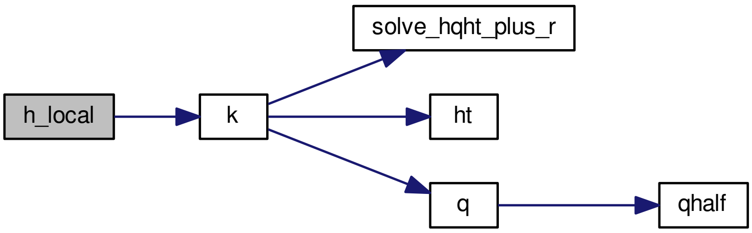


Here is the caller graph for this function:

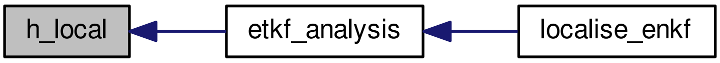


6.10.1.2 subroutine h_local (integer, intent(in) *num_hor,* integer, intent(in) *num_ver,* integer, intent(in) *this_hor,* integer, intent(in) *this_ver,* integer, intent(in) *boundary,* integer, intent(in) *nrhs,* integer, intent(in) *stateDim,* real(kind=rk), dimension(statedim,nrhs), intent(in) *x,*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

Here is the call graph for this function:

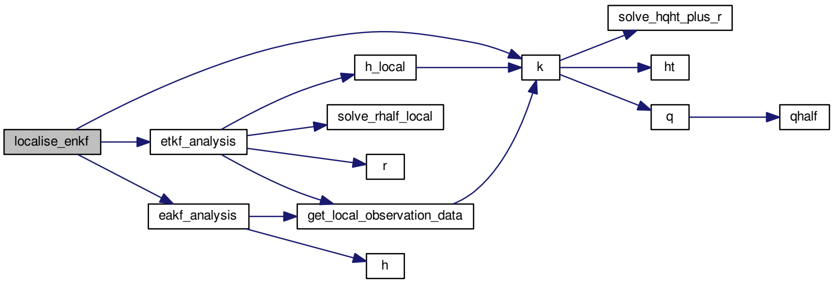


Here is the caller graph for this function:



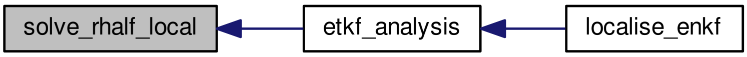
6.10.1.3 subroutine localise_enkf (integer, intent(in) *enkf_analysis*)

Here is the call graph for this function:



6.10.1.4 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*)

Here is the caller graph for this function:



6.11 src/filters/equivalent_weights_step.f90 File Reference

Functions/Subroutines

- subroutine equal_weight_filter

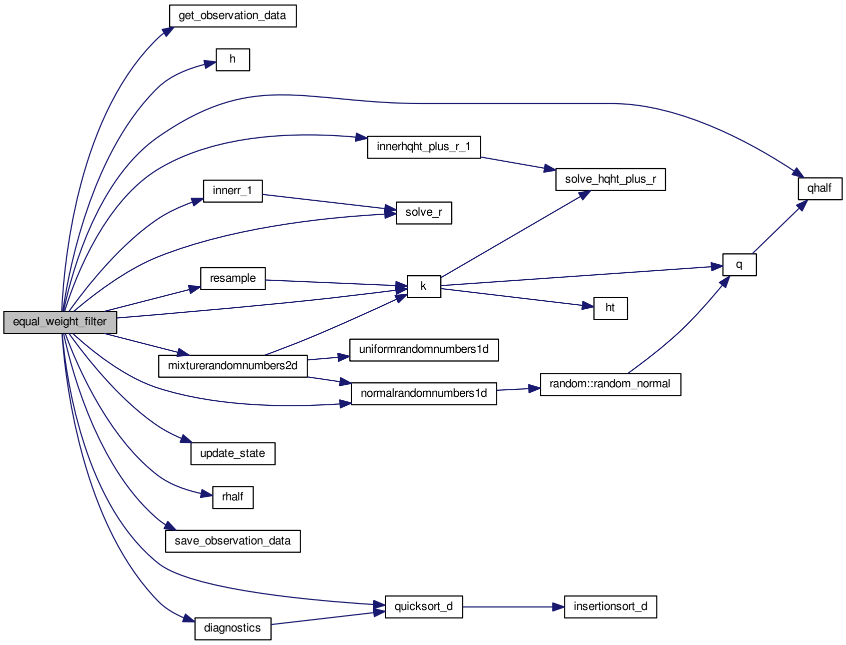
    subroutine to do the equivalent weights step

6.11.1 Function/Subroutine Documentation

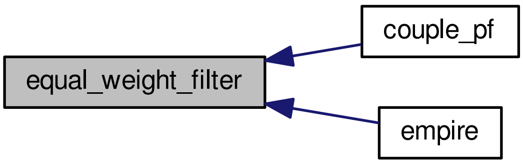
6.11.1.1 subroutine equal_weight_filter ( )

subroutine to do the equivalent weights step

Here is the call graph for this function:



Here is the caller graph for this function:



6.12 src/filters/etkf_analysis.f90 File Reference

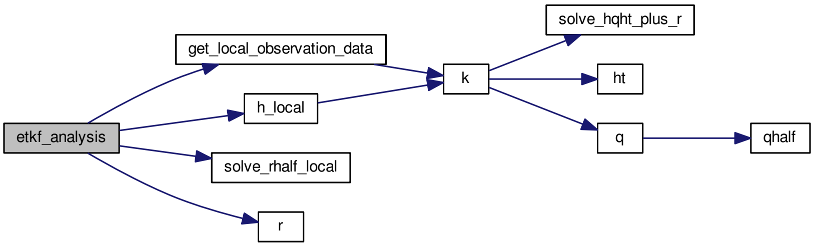
Functions/Subroutines

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

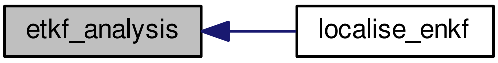
6.12.1 Function/Subroutine Documentation

6.12.1.1 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*)

Here is the call graph for this function:



Here is the caller graph for this function:



6.13 src/filters/proposal_filter.f90 File Reference

Functions/Subroutines

- subroutine proposal_filter

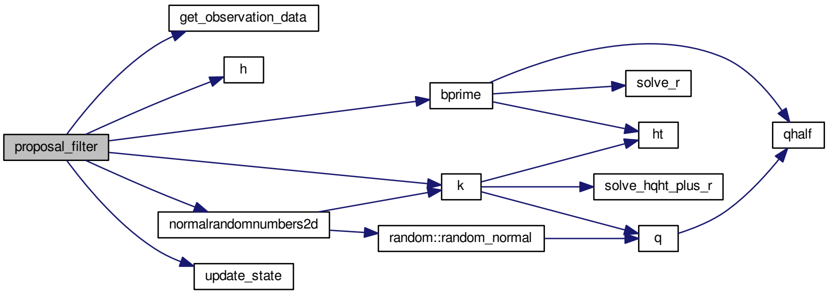
  Subroutine to perform nudging in the proposal step of EWPF.

6.13.1 Function/Subroutine Documentation

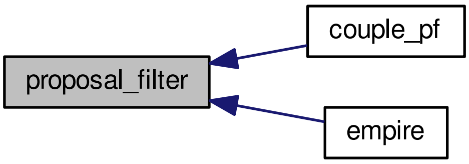
6.13.1.1 subroutine proposal_filter ( )

Subroutine to perform nudging in the proposal step of EWPF.

Here is the call graph for this function:



Here is the caller graph for this function:



6.14 src/filters/sir_filter.f90 File Reference

Functions/Subroutines

- subroutine sir_filter

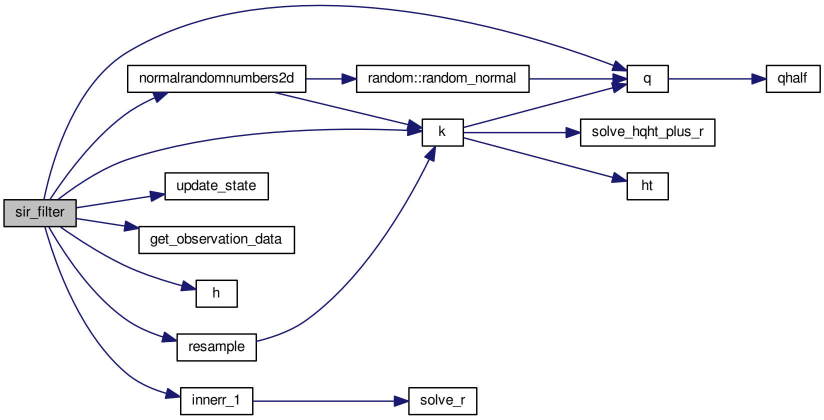
  Subroutine to perform SIR filter (Sequential Importance Resampling)

6.14.1 Function/Subroutine Documentation

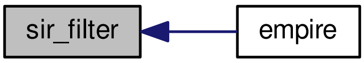
6.14.1.1 subroutine sir_filter ( )

Subroutine to perform SIR filter (Sequential Importance Resampling)

Here is the call graph for this function:



Here is the caller graph for this function:



6.15 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

- subroutine check_scaling (x, fx, b, scales)

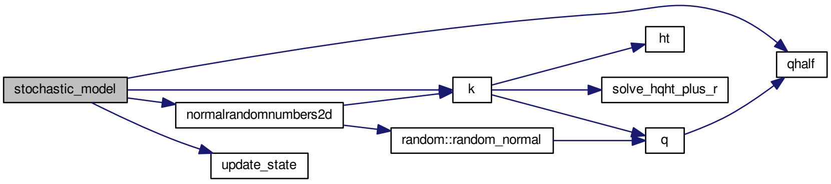
6.15.1 Function/Subroutine Documentation

6.15.1.1 subroutine check_scaling (real(kind=rk), dimension(state_dim), intent(in) *x*, real(kind=rk), dimension(state_dim), intent(in) *fx*, real(kind=rk), dimension(state_dim), intent(in) *b*, real(kind=rk), dimension(9), intent(inout) *scales*)

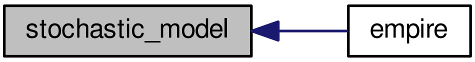
6.15.1.2 subroutine stochastic_model ( )

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.16 src/operations/gen_rand.f90 File Reference

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.

6.16.1 Function/Subroutine Documentation

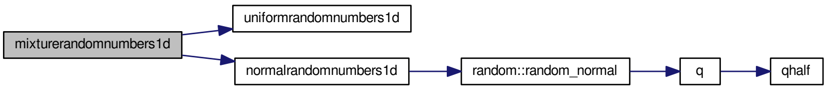
6.16.1.1 subroutine mixturerandomnumbers1d (real(kind=kind(1.0d0)), intent(in) *mean,* real(kind=kind(1.0d0)), intent(in) *stdev,* real(kind=kind(1.0d0)), intent(in) *ufac,* real(kind=kind(1.0d0)), intent(in) *epsi,* integer, intent(in) *n,* real(kind=kind(1.0d0)), dimension(n), intent(out) *phi,* logical,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

Here is the call graph for this function:



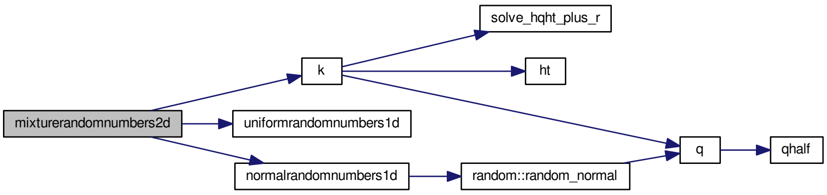
6.16.1.2 subroutine mixturerandomnumbers2d (real(kind=kind(1.0d0)), intent(in) *mean*, real(kind=kind(1.0d0)), intent(in) *stdev*, real(kind=kind(1.0d0)), intent(in) *ufac*, real(kind=kind(1.0d0)), intent(in) *epsi*, integer, intent(in) *n*, integer, intent(in) *k*, real(kind=kind(1.0d0)), dimension(n,k),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

generate two dimensional vector, each drawn from mixture density

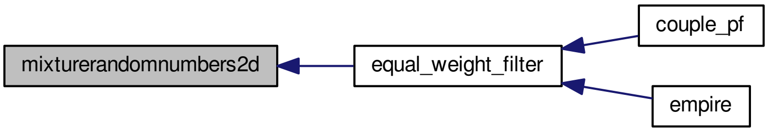
Parameters

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	*n*	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

Here is the call graph for this function:



Here is the caller graph for this function:



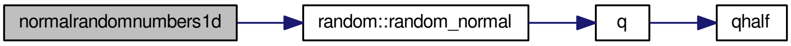
6.16.1.3 subroutine normalrandomnumbers1d (real(kind=rk), intent(in) *mean*, real(kind=rk), intent(in) *stdev*, integer, intent(in) *n*, real(kind=rk), dimension(n), intent(out) *phi*)

generate one dimension of Normal random numbers

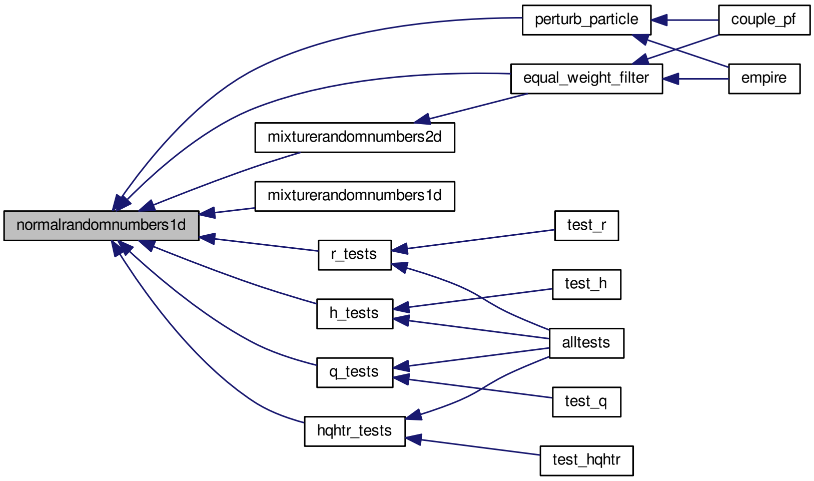
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



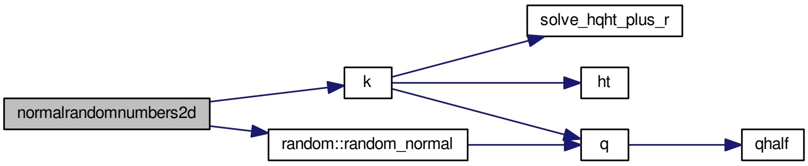
6.16.1.4 subroutine normalrandomnumbers2d (real(kind=rk), intent(in) *mean,* real(kind=rk), intent(in) *stdev,* integer, intent(in) *n,* integer, intent(in) *k,* real(kind=rk), dimension(n,k), intent(out) *phi*)

generate two dimensional Normal random numbers

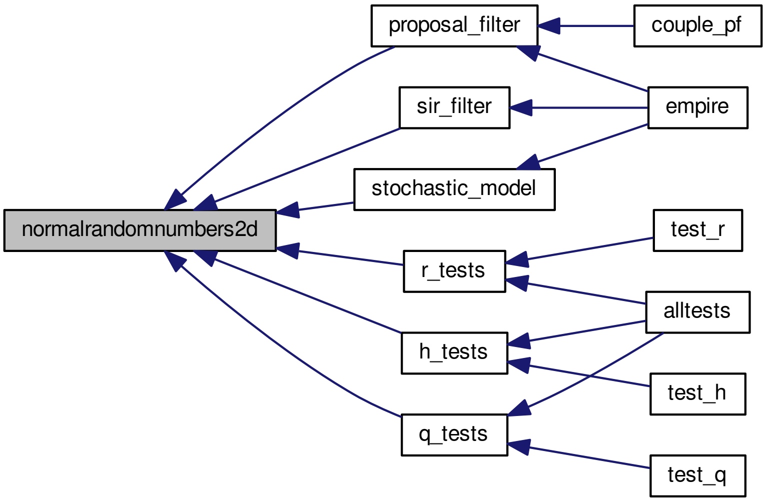
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



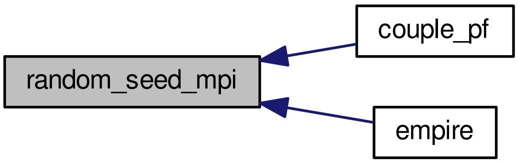
6.16.1.5 subroutine random_seed_mpi (integer, intent(in) *pfid*)

Subroutine to set the random seed across MPI threads.

Parameters

in	*pfid*	The process identifier of the MPI process

Here is the caller graph for this function:



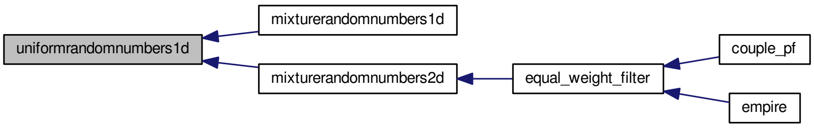
6.16.1.6 subroutine uniformrandomnumbers1d (real(kind=rk), intent(in) *minv*, real(kind=rk), intent(in) *maxv*, integer, intent(in) *n*, real(kind=rk), dimension(n), intent(out) *phi*)

generate one dimension of uniform random numbers

Parameters

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

Here is the caller graph for this function:



6.17 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 innerr_1 (y, w)

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

  subroutine to compute the inner product with $(HQH^T + R)^{-1}$
- subroutine bprime (y, x, QHtR_1y, normaln, betan)

  subroutine to calculate nudging term and correlated random errors efficiently

6.17.1 Function/Subroutine Documentation

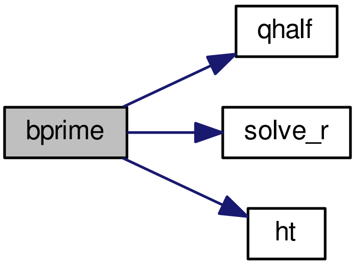
6.17.1.1 subroutine bprime (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y*, real(kind=rk), dimension(state_dim,pf%count), intent(out) *x*, real(kind=rk), dimension(state_dim,pf%count), intent(out) *QHtR_1y*, real(kind=rk), dimension(state_dim,pf%count), intent(in) *normaln*, real(kind=rk),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

subroutine to calculate nudging term and correlated random errors efficiently

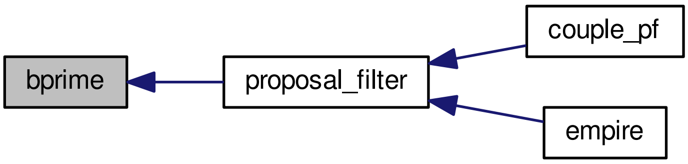
Parameters

in	*y*	(obs_dim,pf%count) vectors of innovations $y-H(x^{n-1})$
out	*x*	(state_dim,pf%count) vectors of $\rho Q^{\frac{1}{2}}H^TR^{-1}[y-H(x^{n-1})]$
out	*QHtR_1y*	(state_dim,pf%count) vectors of $\rho QH^TR^{-1}[y-H(x^{n-1})]$
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) $\sim \mathcal{N}(0,Q)$

Here is the call graph for this function:



Here is the caller graph for this function:



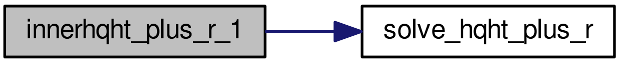
6.17.1.2 subroutine innerhqht_plus_r_1 (real(kind=rk), dimension(obs_dim), intent(in) *y*, real(kind=rk), intent(out) *w*)

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

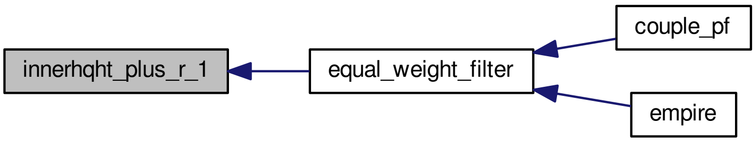
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



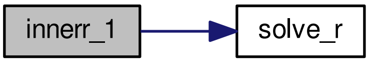
6.17.1.3 subroutine innerr_1 (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y*, real(kind=rk), dimension(pf%count), intent(out) *w*)

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

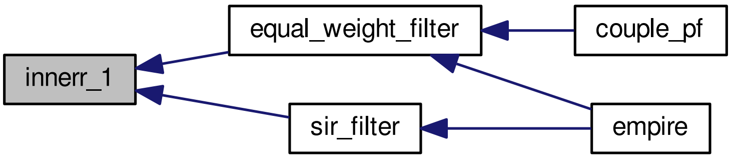
Parameters

in	*y*	multiple vectors in observation space (pf%count of them)
out	*w*	multiple scalars (pf%count) where w(i) has the value $y(:,i)^T R^{-1} y(:,i)$

Here is the call graph for this function:



Here is the caller graph for this function:



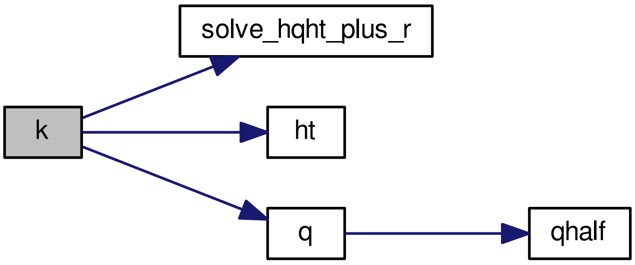
6.17.1.4 subroutine k (real(kind=rk), dimension(obs_dim,pf%count), intent(in) *y*, real(kind=rk), dimension(state_dim,pf%count), intent(out) *x*)

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

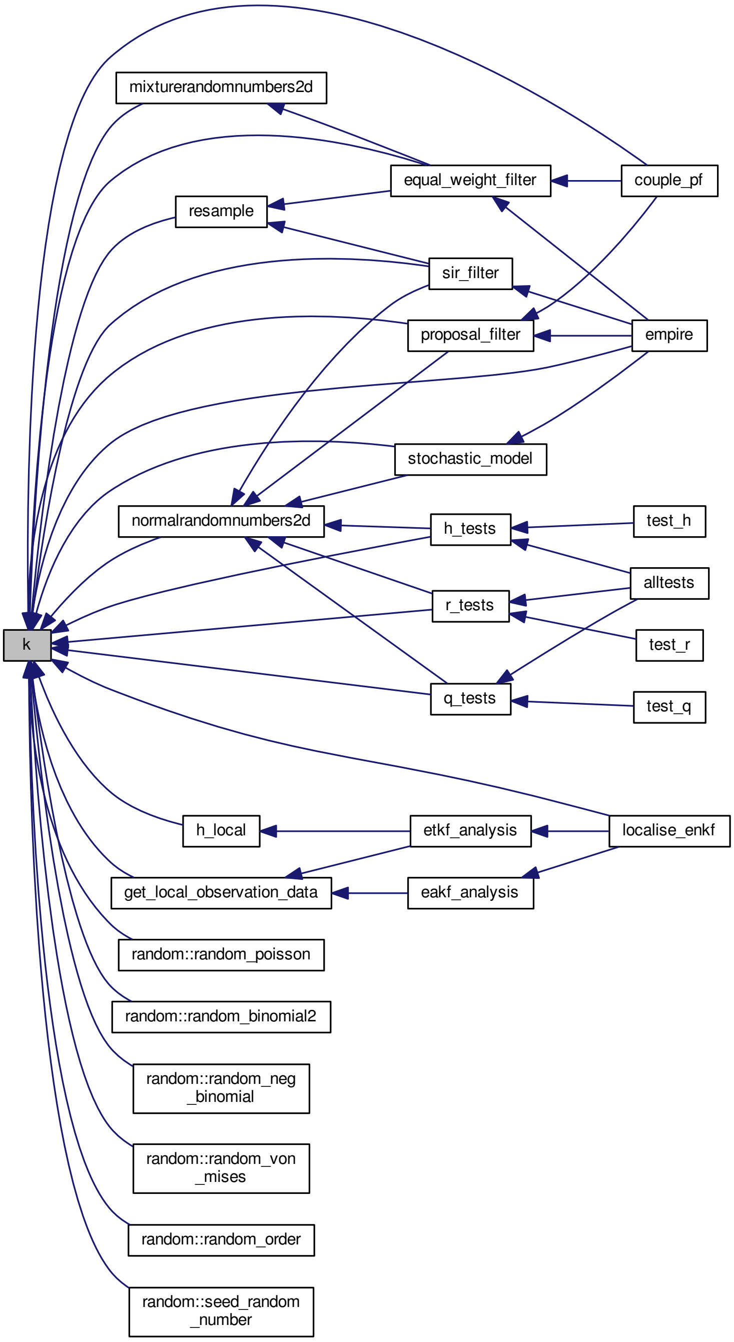
Parameters

in	*y*	vector in observation space
out	*x*	vector in state space

Here is the call graph for this function:



Here is the caller graph for this function:



6.18 src/operations/perturb_particle.f90 File Reference

Functions/Subroutines

- subroutine perturb_particle (x)

  Subroutine to perturb state vector with normal random vector drawn from $\mathcal{N}(0,Q)$.

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

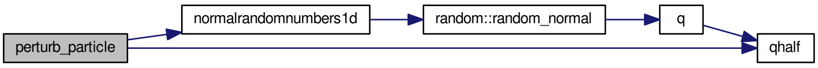
  Subroutine to update the state.

6.18.1 Function/Subroutine Documentation

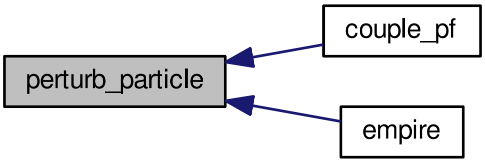
6.18.1.1 subroutine perturb_particle (real(kind=rk), dimension(state_dim), intent(inout) *x*)

Subroutine to perturb state vector with normal random vector drawn from $\mathcal{N}(0,Q)$.

Here is the call graph for this function:



Here is the caller graph for this function:



6.18.1.2 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*)

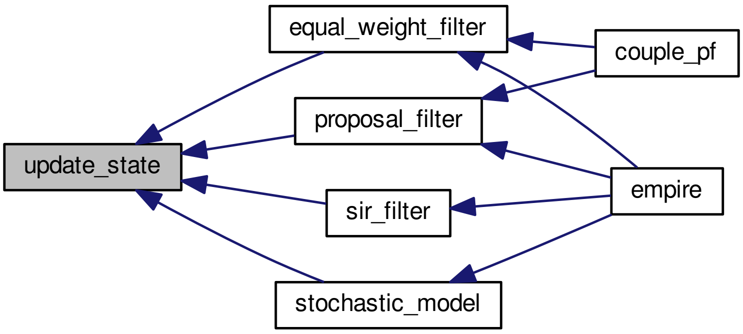
Subroutine to update the state.

This can be changed for the specific model if it needs to be

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

Here is the caller graph for this function:



6.19 src/operations/resample.f90 File Reference

Functions/Subroutines

- subroutine resample

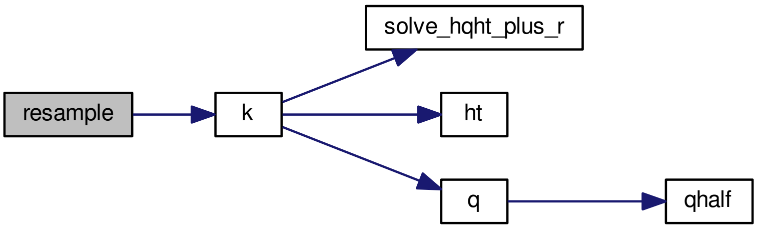
Subroutine to perform Universal Importance Resampling.

6.19.1 Function/Subroutine Documentation

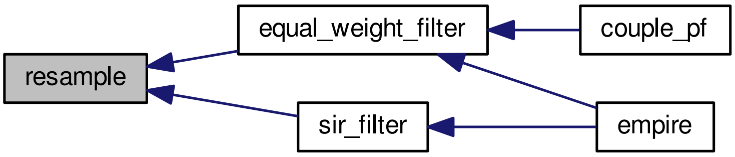
6.19.1.1 subroutine resample ( )

Subroutine to perform Universal Importance Resampling.

Here is the call graph for this function:



Here is the caller graph for this function:



6.20 src/tests/alltests.f90 File Reference

Functions/Subroutines

- program alltests

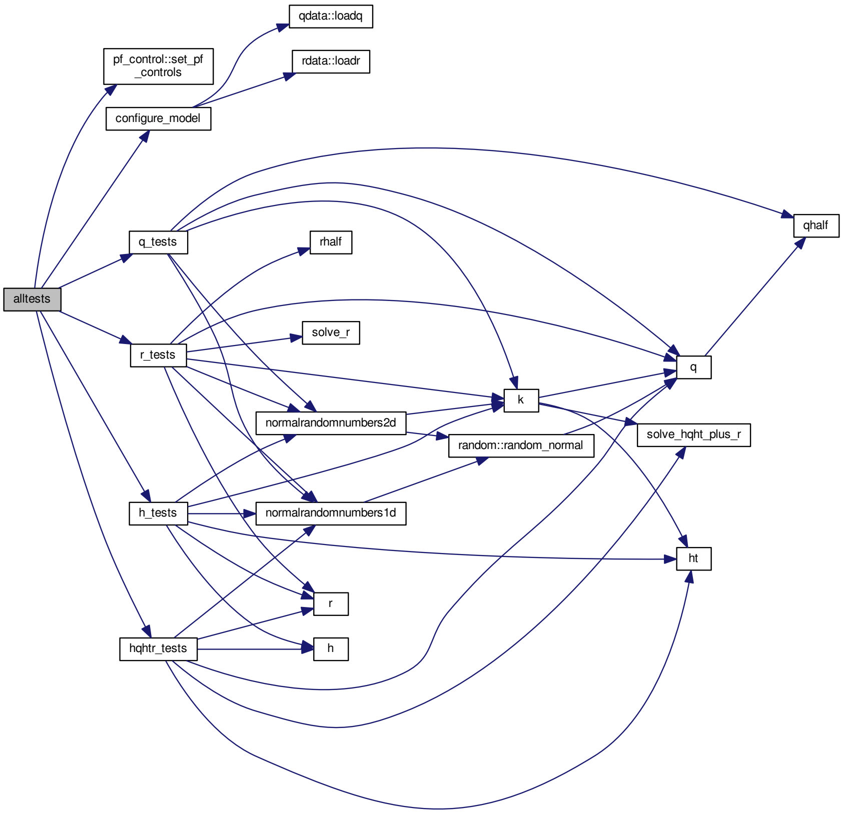
  program to run all tests of user specific functions

6.20.1 Function/Subroutine Documentation

6.20.1.1 program alltests ( )

program to run all tests of user specific functions

Here is the call graph for this function:



6.21 src/tests/test_h.f90 File Reference

Functions/Subroutines

- program test_h

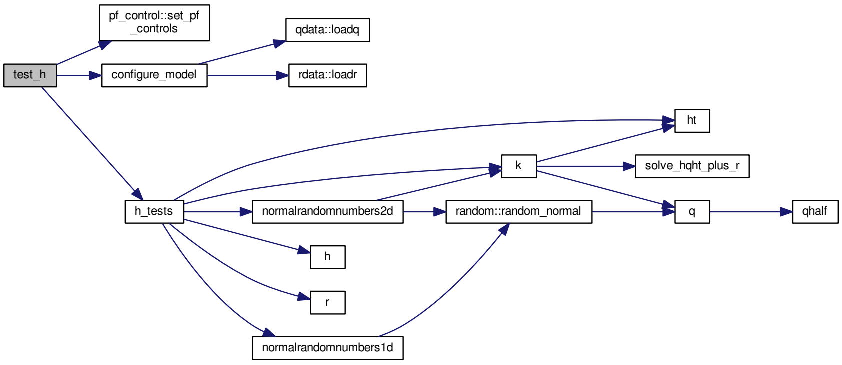
    program to run tests of user supplied observation operator

6.21.1 Function/Subroutine Documentation

6.21.1.1 program test_h ( )

program to run tests of user supplied observation operator

Here is the call graph for this function:



6.22 src/tests/test_hqhtr.f90 File Reference

Functions/Subroutines

- program test_hqhtr

    program to run tests of user supplied linear solve

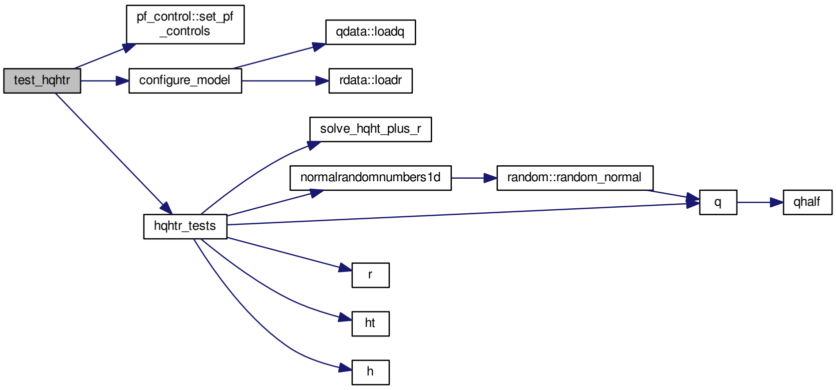
6.22.1 Function/Subroutine Documentation

6.22.1.1 program test_hqhtr ( )

program to run tests of user supplied linear solve

$(HQH^T + R)^{-1}$

Here is the call graph for this function:



6.23 src/tests/test_q.f90 File Reference

Functions/Subroutines

- program test_q

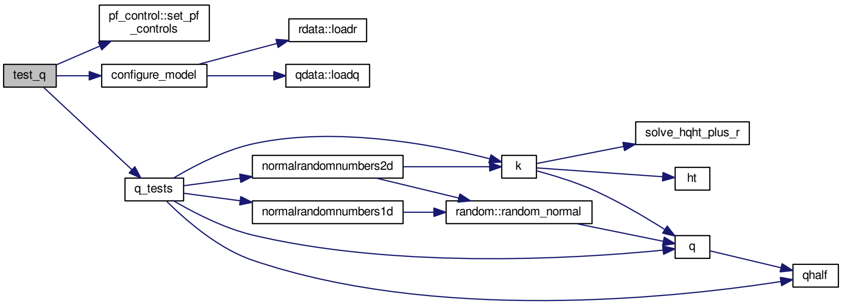
  program to run tests of user supplied model error covariance matrix

6.23.1 Function/Subroutine Documentation

6.23.1.1 program test_q ( )

program to run tests of user supplied model error covariance matrix

Here is the call graph for this function:



6.24 src/tests/test_r.f90 File Reference

Functions/Subroutines

- program test_r

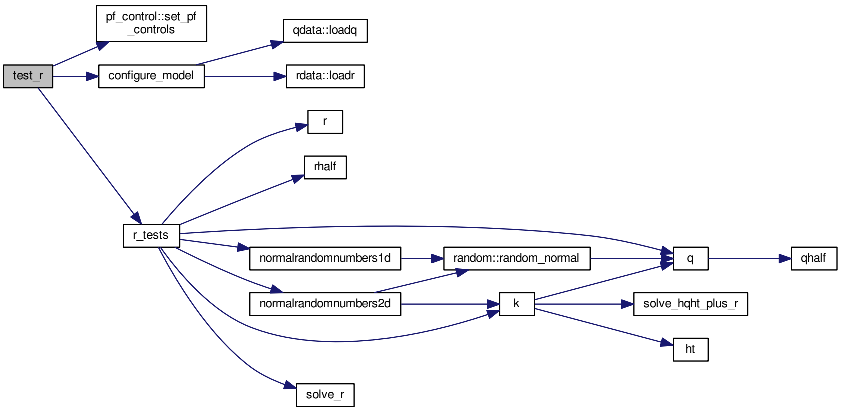
  program to run all tests of user supplied observation error covariance matrix/

6.24.1 Function/Subroutine Documentation

6.24.1.1 program test_r ( )

program to run all tests of user supplied observation error covariance matrix/

Here is the call graph for this function:



6.25 src/tests/tests.f90 File Reference

Functions/Subroutines

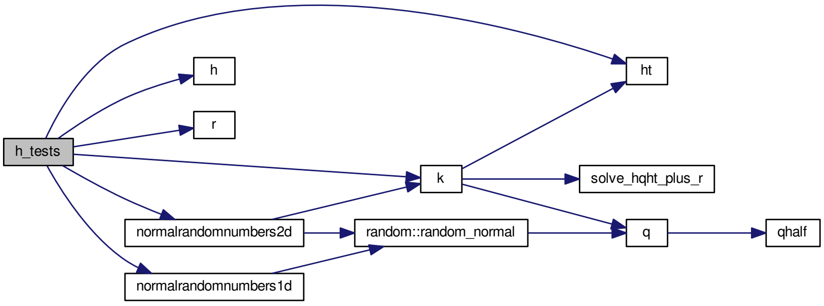
- subroutine h_tests ()
- subroutine r_tests ()
- subroutine q_tests ()
- subroutine hqhtr_tests ()

6.25.1 Function/Subroutine Documentation

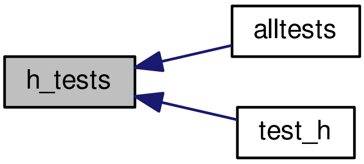
6.25.1.1 subroutine h_tests ( )

These are some tests to check that the observation operator is implemented correctly

Here is the call graph for this function:



Here is the caller graph for this function:

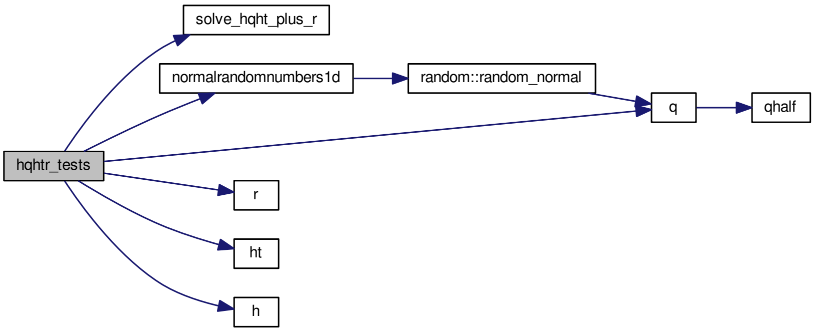


6.25.1.2 subroutine hqhtr_tests ( )

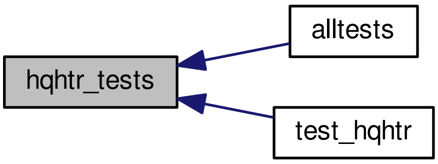
These are some tests to check that the linear solve operator is implemented correctly

This should check the operation $(HQH^T+R)^{-1}$ is working

Here is the call graph for this function:



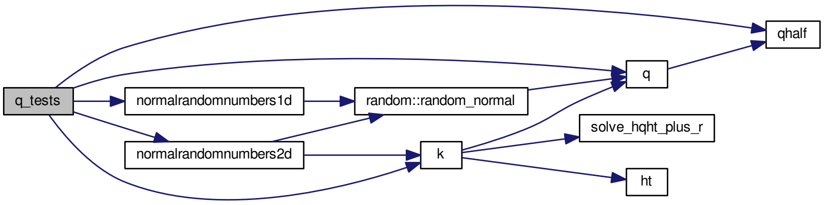
Here is the caller graph for this function:



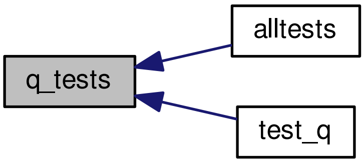
6.25.1.3 subroutine q_tests ( )

These are some tests to check that the model error covariance matrix is implemented correctly

Here is the call graph for this function:



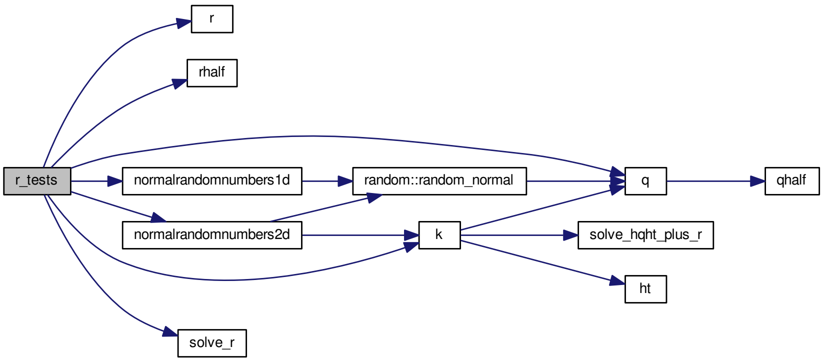
Here is the caller graph for this function:



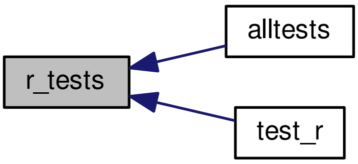
6.25.1.4 subroutine r_tests ( )

These are some tests to check that the observation error covariance matrix is implemented correctly

Here is the call graph for this function:



Here is the caller graph for this function:



6.26 src/utils/comms.f90 File Reference

Data Types

- module comms

  Module containing EMPIRE coupling data.

6.27 src/utils/data_io.f90 File Reference

Functions/Subroutines

- subroutine get_observation_data (y)

  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 save_truth (x)

  Subroutine to save truth to a file
  .

- subroutine output_from_pf

  subroutine to ouput data from the filter

6.27.1 Function/Subroutine Documentation

6.27.1.1 subroutine get_observation_data (real(kind=rk), dimension(obs_dim), intent(out) *y*)

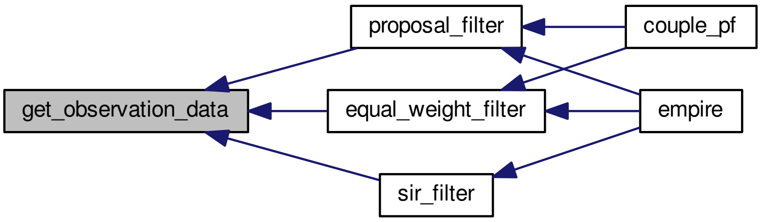
Subroutine to read observation from a file

Uses pftimestep to determine which observation to read.

Parameters

out	*y*	The observation

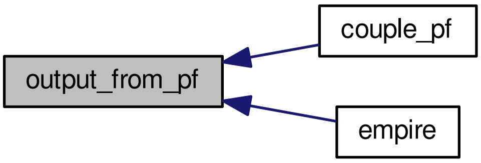
Here is the caller graph for this function:



6.27.1.2 subroutine output_from_pf ( )

subroutine to ouput data from the filter

Here is the caller graph for this function:



6.27.1.3 subroutine save_observation_data (real(kind=rk), dimension(obs_dim), intent(in) *y*)

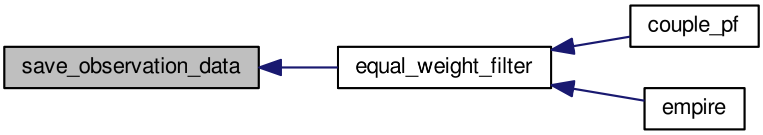
Subroutine to save observation to a file

Uses pftimestep to determine which observation to save.

Parameters

in	*y*	The observation

Here is the caller graph for this function:



6.27.1.4 subroutine save_truth (real(kind=rk), dimension(state_dim), intent(in) *x*)

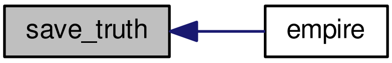
Subroutine to save truth to a file

.

Parameters

in	*x*	The state vector

Here is the caller graph for this function:



6.28 src/utils/diagnostics.f90 File Reference

Functions/Subroutines

- subroutine diagnostics

  Subroutine to give output diagnositics such as rank histograms and trajectories.

- subroutine trajectories

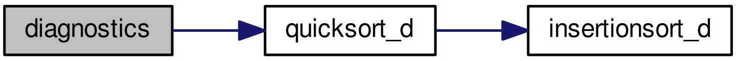
  subroutine to output trajectories

6.28.1 Function/Subroutine Documentation

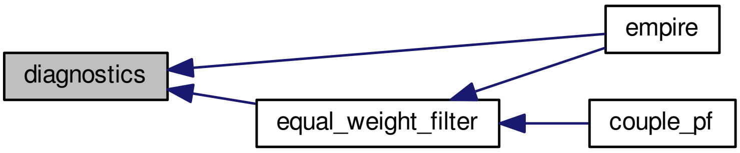
6.28.1.1 subroutine diagnostics ( )

Subroutine to give output diagnositics such as rank histograms and trajectories.

Here is the call graph for this function:



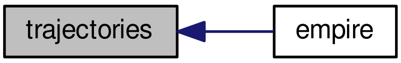
Here is the caller graph for this function:



6.28.1.2 subroutine trajectories ( )

subroutine to output trajectories

Here is the caller graph for this function:



6.29 src/utils/genQ.f90 File Reference

Functions/Subroutines

- subroutine genq

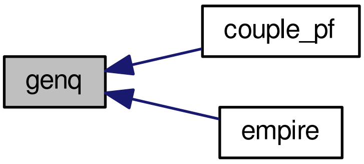
  Subroutine to estimate Q from a long model run.

6.29.1 Function/Subroutine Documentation

6.29.1.1 subroutine genq ( )

Subroutine to estimate Q from a long model run.

Here is the caller graph for this function:



6.30 src/utils/histogram.f90 File Reference

Data Types

- module histogram_data

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

6.31 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

6.31.1 Function/Subroutine Documentation

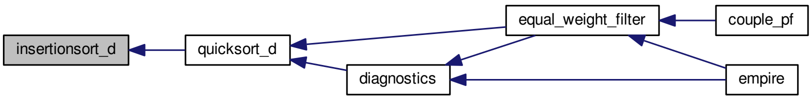
6.31.1.1 subroutine insertionsort_d (real(kind=kind(1.0d0)), dimension(na), intent(inout) *A*, integer, intent(in) *nA*)

subroutine to sort using the insertionsort algorithm

Parameters

`in,out`	*a*	array of doubles to be sorted
`in`	*na*	dimension of array a

Here is the caller graph for this function:



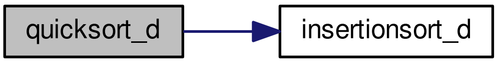
6.31.1.2 recursive subroutine quicksort_d (real(kind=kind(1.0d0)), dimension(na), intent(inout) *a*, integer, intent(in) *na*)

subroutine to sort using the quicksort algorithm

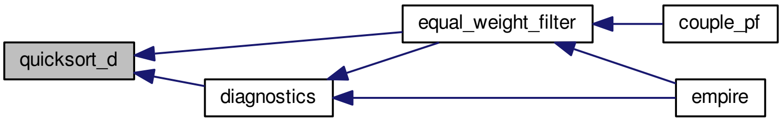
Parameters

`in,out`	*a*	array of doubles to be sorted
`in`	*na*	dimension of array a

Here is the call graph for this function:



Here is the caller graph for this function:



6.32 src/utils/random_d.f90 File Reference

Data Types

- module random

  A module for random number generation from the following distributions:

Index