REHEATFUNQ

Release 2.0.1

Malte J. Ziebarth

Jan 06, 2024

CONTENTS:

1 Installation **3**
- 1.1 Local Install . . . 3
- 1.2 Container/Docker Images . . . 3
- 1.3 Known Issues . . . 5

2 Quickstart **7**
- 2.1 Simple Heat Flow and Anomaly Analysis . . . 7
- 2.2 REHEATFUNQ Paper . . . 8

3 Regional Heat Flow **9**
- 3.1 `reheatfunq.regional` . . . 9

4 Anomaly quantification **19**
- 4.1 `reheatfunq.anomaly` . . . 19

5 RGRDC **27**
- 5.1 `reheatfunq.coverings` . . . 27

6 Heat Flow Data **31**
- 6.1 `reheatfunq.data` . . . 31

7 Resilience Analysis **33**
- 7.1 `reheatfunq.resilience` . . . 33

8 Bibliography **39**

9 Indices and tables **41**

Bibliography **43**

Python Module Index **45**

Index **47**

REHEATFUNQ is a Python package for the analysis of regional aggregate heat flow distributions, and the quantification of heat flow anomalies superposing data drawn from these distributions.

CHAPTER

ONE

INSTALLATION

1.1 Local Install

First, make sure that the libraries and packages listed in **Dependencies** within README.md are installed. Installation might differ per operating system. Most of the Python packages should be available from PyPI.

Afterwards, to install REHEATFUNQ locally, run the following command in the REHEATFUNQ source code root directory

```
pip install --user .
```

Alternatively, if you have not downloaded the source yet, you can run the following command:

```
pip install 'reheatfunq @ git+https://github.com/mjziebarth/REHEATFUNQ'
```

1.2 Container/Docker Images

REHEATFUNQ can also be used within the provided Docker images. The images contain a Jupyter notebook server running as user `reheatfunq`, and all required packages are installed.

Two container images are supplied: `Dockerfile` and `Dockerfile-stable`. The former builds on the `python:slim-bookworm` image and pulls up-to-date dependencies from the web. It is more lightweight, uses considerably less compile time, and can utilize new features of the updated software. On the flip side, this image might not be able to (exactly) reproduce the simulations from the paper if any of the important packages introduces changes to the numerics.

For this reason, `Dockerfile-stable` is provided which uses vendored sources and should stay reproducible in the long term. It builds upon a snapshot of the Debian `slim` image for basic functionality and vendors the relevant source code of the REHEATFUNQ model and its foundations to build a reproducible model.

1.2.1 `Dockerfile`

Two container services are covered in this documentation, Podman and Docker. To build the `Dockerfile` container, run either

```
podman build --format docker -t reheatfunq .
```

or

```
sudo docker build -t 'reheatfunq' .
```

within the repository's root directory (`sudo` may or may not be required depending on the Docker setup).

The Jupyter notebook server is exposed at the container's 8888 port. This port may or may not be free on your system. To run REHEATFUNQ in the container, first identify a free port `XXXX` on your machine. Then, run

```
podman run -p XXXX:8888 reheatfunq
```

or

```
sudo docker run -p XXXX:8888 reheatfunq
```

The container image does not contain all required data to run the analysis of the REHEATFUNQ paper. Most prominently, that includes the `NGHF.csv` of Lucazeau [L2019]. A convenient method to copy this (or other files you wish to copy) to the running container is the Jupyter server file up- and download dialog.

You can shut down the container image by quitting the Jupyter server via the web interface.

1.2.2 `Dockerfile-stable`

This container image requires the sources of the software upon which REHEATFUNQ is built. The combined source code archive of this software is large (the `Dockerfile-stable` starts by bootstrapping the GNU Compiler Collection and successively compiles the Python ecosystem and numeric software) and it is split off this git repository. The archives `vendor-1.3.3.tar.xz` and `vendor-2.0.1.tar.xz` from GFZ Data Services contains the required software.

There are two ways to build the `Dockerfile-stable` image. Since version 2.0.1, the shell script *build-Dockerfile-stable.sh* is available to facilitate the build process. Using this script, the build process should be as simple as

```
bash build-Dockerfile-stable.sh
```

A second way is to build the image manually. This first requires downloading `vendor-1.3.3.tar.xz` and `vendor-2.0.1.tar.xz`. Following the instructions presented therein, extract the `compile` and `wheels` subfolders into the `vendor` directory of this repository.

Then, you can build and run the Docker image as below (you might want to rename the container according to the REHEATFUNQ version you are using—unless stated otherwise, the following versions are compatible with `vendor-1.3.3.tar.xz` and `vendor-2.0.1.tar.xz`):

```
podman build --format docker -f Dockerfile-stable -t reheatfunq-2.0.1-stable
podman run -p XXXX:8888 reheatfunq-2.0.1-stable
```

or

```
sudo docker build -f Dockerfile-stable -t 'reheatfunq-2.0.1-stable' .
sudo docker run -p XXXX:8888 reheatfunq-2.0.1-stable
```

Nearly all of the dependencies of this container are contained in `vendor-1.3.3.tar.xz` and `vendor-2.0.1.tar.xz` so that this image should build reproducibly in the long-term. Nevertheless, the Debian snapshot used as a base image might be unavailable at some point in the future of this writing. In this case, it should be possible to swap the base image to another linux without great impact. For the purpose of base image agnosticism, the container image rebuilds `gcc` and installs libraries to the `/sci` directory.

In case that swapping the base image is neccessary but does not work out of the box, it is likely that the initial user setup or the installation of build tools to bootstrap `gcc` has to be adjusted.

1.3 Known Issues

1.3.1 Cython 3.0.4 compile failure (REHEATFUNQ v1.4.0)

With Cython version 3.0.4 (potentially also other versions), REHEATFUNQ v1.4.0 may fail to install locally with a (fairly extensive) error message that boils down to the following error:

```
reheatfunq/coverings/rdisks.pyx:235:27: Cannot assign type 'iterator' to 'const_iterator'
```

On Cython 3.0.4, this issue can be fixed by editing line 213 of the file `reheatfunq/coverings/rdisks.pyx` from

```
cdef unordered_map[vector[cbool],size_t].iterator it
```

to

```
cdef unordered_map[vector[cbool],size_t].const_iterator it
```

Local install should now proceed normally.

CHAPTER TWO

QUICKSTART

2.1 Simple Heat Flow and Anomaly Analysis

A quick analysis can be done using just a few commands. This assumes that the heat flow data has already been loaded to the variables `hf_x`, `hf_y`, and `hf_mWm2`, where the former are the data point coordinates in a map-projected coordinate system and the latter is an array of the heat flow data (in $\mathrm{mWm^{-2}}$). For instance, the data could be loaded from the NGHF [L2019] using the *`reheatfunq.data`* module. All arrays are assumed to be one-dimensional continuous numpy arrays.

First, perform the necessary imports:

```python
import numpy as np
import matplotlib.pyplot as plt
from reheatfunq.regional import (default_prior,
                                  HeatFlowPredictive)
from reheatfunq.anomaly import (HeatFlowAnomalyPosterior,
                                 AnomalyLS1980)
```

Obtain the default *`GammaConjugatePrior`* and compute the predictive cumulative distribution function, that is, the estimate of the regional aggregate heat flow distribution:

```python
gcp = default_prior()
predictive = HeatFlowPredictive(hf_mWm2, hf_x, hf_y, gcp,
                                 dmin=20e3)

qplt = np.linspace(30, 90)
cdf = predictive.cdf(qplt)
```

Now `qplt` is a range of heat flow values from $30\,\mathrm{mWm^{-2}}$ to $90\,\mathrm{mWm^{-2}}$, and `cdf` holds the corresponding values of the estimated heat flow CDF.

2.1.1 Fault-generated heat flow anomaly

Let's assume that the variable `fault_trace` is a `(N,2)`-shaped NumPy array that holds the consecutive coordinates of a fault trace `[(x0,y0), ..., (xN,yN)]`. We would like to investigate the strength of the potential heat flow anomaly generated by this fault. Let's assume furthermore that we are content with approximating the heat conduction from that fault by the *`AnomalyLS1980`* class (implying that in the relevant vicinity of the data the fault is vertical and straight, and heat transport conductive, and that heat production increases linearly with depth). Finally, let `D` be the uniform depth of the fault in meters.

Then we can use the following code to quantify the heat-generating power on the fault through the heat flow anomaly strength:

```
anomaly = AnomalyLS1980(fault_trace, D)
post = HeatFlowAnomalyPosterior(hf_mWm2, hf_x, hf_y, anomaly,
                                gcp)

P_H = np.linspace(0, post.PHmax, 200)
pdf_P_H = post.pdf(P_H)
tail_P_H = post.tail(P_H)
```

The parameter `post.PHmax` is the maximum heat production power on the fault, that is, for all greater powers the posterior has zero probability density. The variables `pdf_P_H` and `tail_P_H` now hold the marginal posterior density and tail distribution, respectively, of the heat production power P_H.

A readymade Jupyter notebook for this analysis can be found in jupyter/Quickstart.ipynb.

2.2 REHEATFUNQ Paper

To repeat the analysis performed in the REHEATFUNQ paper, you can use the notebooks prefixed **01** to **06** in the jupyter/REHEATFUNQ/ folder.

CHAPTER

THREE

REGIONAL HEAT FLOW

3.1 reheatfunq.regional

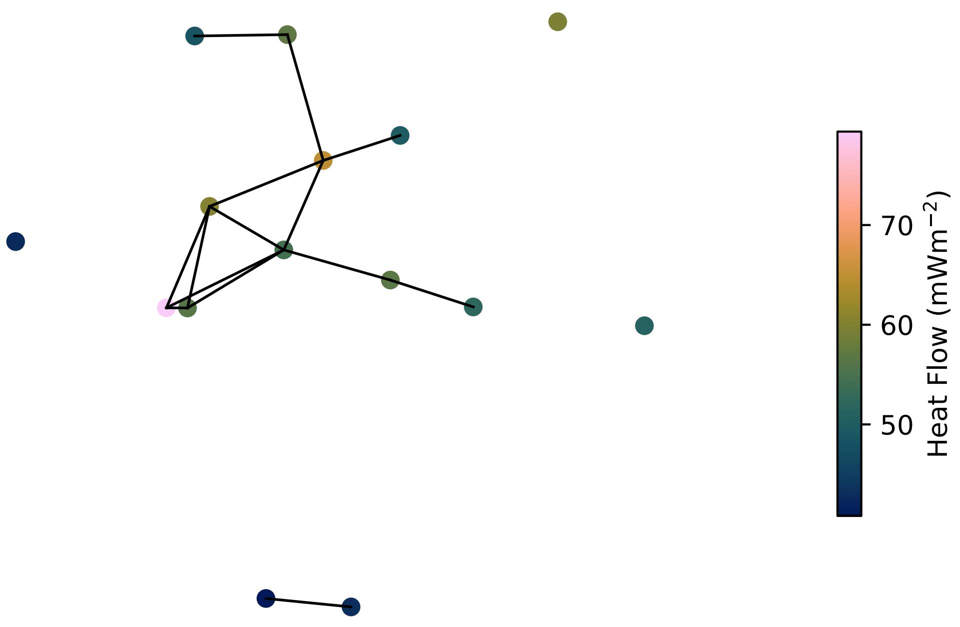
The *reheatfunq.regional* module contains functionality to analyze regional aggregate heat flow distributions using the *GammaConjugatePrior* and *HeatFlowPredictive* classes. The workflow for regional aggregate heat flow analysis using REHEATFUNQ consists of the following steps:

1. Define the d_min (e.g. $d_\mathrm{min} = 20\,\mathrm{km}$)
2. Define the conjugate prior to use. Obtain a *GammaConjugatePrior* instance (e.g. using the REHEATFUNQ default from *default_prior()*).
3. Compute the posterior predictive heat flow distribution using the *HeatFlowPredictive* class. This class performs the bootstrapped updating of the gamma conjugate prior over the set of d_min-conforming subsets of the heat flow data.

Exemplarily, the following code summarizes the analysis. First, we generate some toy heat flow data following a gamma distribution:

```
import numpy as np
rng = np.random.default_rng(123920)
alpha = 53.3
q = rng.gamma(alpha, size=15)
x = 100e3 * (rng.random(15) - 0.5)
y = 100e3 * (rng.random(15) - 0.5)
```

The resulting synthetic heat flow data should look like this:

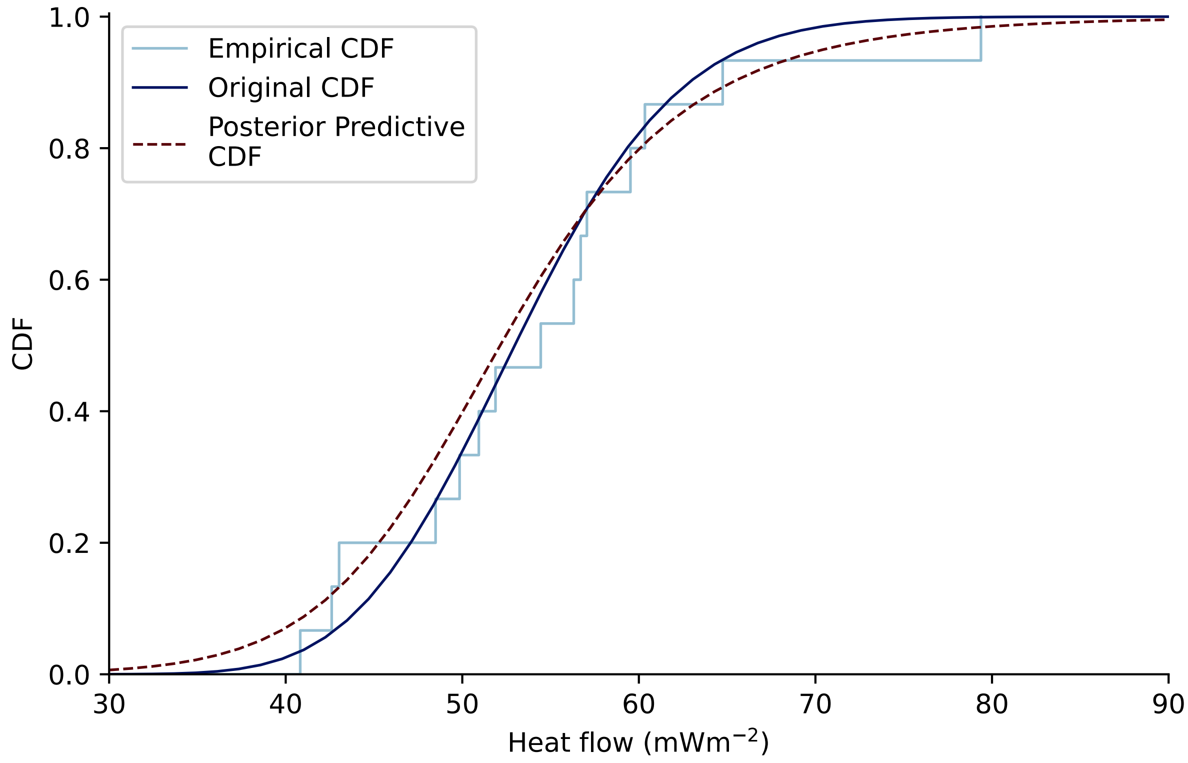


Next, the *GammaConjugatePrior* and *HeatFlowPredictive* classes can be used to evaluate the regional aggregate heat flow distribution from this data:

```
from reheatfunq.regional import (GammaConjugatePrior,
                                 default_prior,
                                 HeatFlowPredictive)
gcp = default_prior()
predictive = HeatFlowPredictive(q, x, y, gcp, dmin=20e3)

qplt = np.linspace(35, 85)
cdf = predictive.cdf(qplt)
```

The posterior predictive CDF is a broadened compared to the original CDF owing to the finite sample size and averaging over alternating data points for pairs within exclusory distance:



This image might vary slightly due to the non-fixed random number generator in the *HeatFlowPredictive* class.

A detailed use of the regional aggregate heat flow distribution estimation can be found in the Jupyter notebook jupyter/REHEATFUNQ/06-Heat-Flow-Analysis.ipynb.

class GammaConjugatePrior(*p*, *s*, *n*, *v*, *lp=None*, *amin=1.0*)

Gamma conjugate prior by Miller [Miller1980].

Parameters

- **p** (*float | None*) – The parameter p of the gamma conjugate prior. Can be seen as the initial product of heat flow values. Alternatively, $\ln p$ can be specified through the `lp` parameter when passing `None` as argument for `p`..
- **s** (*float*) – The parameter s of the gamma conjugate prior. Can be seen as the initial sum of heat flow values.
- **n** (*float*) – The parameter n of the gamma conjugate prior. For normalization, $n \geq v$ needs to be fulfilled.
- **v** (*float*) – The parameter v of the gamma conjugate prior. For normalization, $n \geq v$ needs to be fulfilled.
- **lp** (*float | None*) – The natural logarithm of the parameter p. An alternative way to specify p.
- **amin** (*float*) – The minimum α for which the prior is defined. Has to be non-negative.

`kullback_leibler`(*other*, *amin=1.0*, *epsabs=0.0*, *epsrel=1e-10*)

Compute the Kullback-Leibler divergence to another gamma conjugate prior.

Parameters

- **other** (GammaConjugatePrior | *Iterable*[GammaConjugatePrior]) – Other gamma conjugate prior(s).
- **epsabs** (*float, optional*) – Absolute tolerance parameter passed to the quadrature routines.
- **epsrel** (*float, optional*) – Relative tolerance parameter passed to the quadrature routines

Returns

KL – The maximum of the Kullback-Leibler divergences from this reference prior PDF to ther `other` PDF(s).

Return type

float

`log_likelihood`(*a*, *b*)

Evaluate the log-likelihood given a set of gamma parameters $\{(\alpha_i, \beta_i) : i = 1, ..., N\}$.

Parameters

- **a** (*array_like*) – Set of gamma distribution shape parameters a. Has to be of the same shape as `b`.
- **b** (*array_like*) – Set of gamma distribution scale parameters b. Has to be of the same shape as `a`.

Returns

p – The logarithm of the evaluated prior probability of the parameter pairs $\{(\alpha_i, \beta_i)\}$.

Return type

array_like

`log_probability`(*a*, *b*)

Evaluate the logarithm of the probability at parameter points.

Parameters

- **a** (*array_like*) – Set of gamma distribution shape parameters a. Has to be of the same shape as `b`.
- **b** (*array_like*) – Set of gamma distribution scale parameters b. Has to be of the same shape as `a`.

Returns

p – The logarithm of the evaluated prior probability of the parameter pairs $\{(\alpha_i, \beta_i)\}$.

Return type

array_like

`static maximum_likelihood_estimate`(*a*, *b*, *p0=1.0*, *s0=1.0*, *n0=1.5*, *v0=1.0*, *nv_surplus_min=0.04*, *vmin=0.1*, *amin=1.0*, *epsabs=0.0*, *epsrel=1e-10*)

Compute the maximum likelihood estimate of the gamma conjugate prior (GCP) given a set of gamma distribution parameters $\{(\alpha_i, \beta_i) : i = 1, ..., N\}$.

Parameters

- **a** (*array_like*) – Set of gamma distribution shape parameters a. Has to be of the same shape as `b`.
- **b** (*array_like*) – Set of gamma distribution scale parameters b. Has to be of the same shape as `a`.
- **p0** (*float, optional*) – Initial guess for the GCP parameter p.
- **s0** (*float, optional*) – Initial guess for the GCP parameter s.
- **n0** (*float, optional*) – Initial guess for the GCP parameter n.
- **v0** (*float, optional*) – Initial guess for the GCP parameter v.
- **nv_surplus_min** (*float, optional*) – Ensures that `n >= v * (1 + nv_surplus_min)`.
- **amin** (*float, optional*) – The minimum α for which the prior is defined. Has to be non-negative.
- **epsabs** (*float, optional*) – Absolute tolerance parameter passed to the optimization algorithm.
- **epsrel** (*float, optional*) – Relative tolerance parameter passed to the optimization algorithm.

Returns
gcp – The gamma conjugate prior with optimized parameters.

Return type
GammaConjugatePrior

static minimum_surprise_estimate(*hf_samples*, *pmin=1.0*, *pmax=100000.0*, *smin=0.0*, *smax=1000.0*, *vmin=0.02*, *vmax=1.0*, *nv_surplus_min=1e-08*, *nv_surplus_max=2.0*, *amin=1.0*, *shgo_kwargs={}*, *cache=None*, *verbose=False*, *return_shgo_result=False*)

Compute the parameter estimate of the gamma conjugate prior (GCP) that minimizes the maximum Kullback-Leibler divergence between the GCP and any of the gamma distribution likelihood computed over a set of heat flow data sets.

Parameters

- **hf_samples** (*Iterable[array_like]*) – A set of heat flow data sets.
- **pmin** (*float, optional*) – Minimum value for the GCP p parameter.
- **pmax** (*float, optional*) – Maximum value for the GCP p parameter.
- **smin** (*float, optional*) – Minimum value for the GCP s parameter.
- **smax** (*float, optional*) – Maximum value for the GCP s parameter.
- **vmin** (*float, optional*) – Minimum value for the GCP v parameter.
- **vmax** (*float, optional*) – Maximum value for the GCP v parameter.
- **nv_surplus_min** (*float, optional*) – Lower bound for the GCP n parameter depending on the v parameter. Ensures that `n >= v * (1 + nv_surplus_min)`.
- **nv_surplus_max** (*float, optional*) – Upper bound for the GCP n parameter depending on the v parameter. Ensures that `n <= v * (1 + nv_surplus_max)`.
- **amin** (*float, optional*) – The minimum α for which the prior is defined. Has to be non-negative.

- **shgo_kwargs** (*dict, optional*) – Additional parameters to pass to the `scipy.optimize.shgo()` global optimizer.
- **cache** (*GCPMSECache, optional*) – A cache for the evaluation of the Kullback-Leiber distance cost function. Can be used to speed up the SHGO optimization when it samples a large number of times.
- **verbose** (*bool, optional*) – If `True`, print some additional progress information.
- **return_shgo_result** (*bool, optional*) – If `True`, return the `scipy.optimize.OptimizeResult` from the SHGO optimization.

Returns
gcp – The gamma conjugate prior with optimized parameters.

Return type
GammaConjugatePrior

`static` **`minimum_surprise_estimate_cache`**(*hf_samples*, *amin=1.0*)

Yields a cache for calling the `minimum_surprise_estimate()` method multiple times with the same parameters but different optimizer settings.

Parameters

- **hf_samples** (*Iterable[array_like]*) – The set of heat flow samples to which the Kullback-Leibler distance from various points of the parameter space is going to be computed.
- **amin** (*float, optional*) – The minimum α for which the prior is defined. Has to be non-negative.

Returns
cache – The cache object.

Return type
GCPMSECache

`static` **`mle`**(*a*, *b*, *p0=1.0*, *s0=1.0*, *n0=1.5*, *v0=1.0*, *nv_surplus_min=0.04*, *vmin=0.1*, *amin=1.0*, *epsabs=0.0*, *epsrel=1e-10*)

Compute the maximum likelihood estimate of the gamma conjugate prior (GCP) given a set of gamma distribution parameters $\{(\alpha_i, \beta_i) : i = 1, ..., N\}$.

Parameters

- **a** (*array_like*) – Set of gamma distribution shape parameters a. Has to be of the same shape as `b`.
- **b** (*array_like*) – Set of gamma distribution scale parameters b. Has to be of the same shape as `a`.
- **p0** (*float, optional*) – Initial guess for the GCP parameter p.
- **s0** (*float, optional*) – Initial guess for the GCP parameter s.
- **n0** (*float, optional*) – Initial guess for the GCP parameter n.
- **v0** (*float, optional*) – Initial guess for the GCP parameter v.
- **nv_surplus_min** (*float, optional*) – Ensures that `n >= v * (1 + nv_surplus_min)`.
- **amin** (*float, optional*) – The minimum α for which the prior is defined. Has to be non-negative.

- **epsabs** (*float, optional*) – Absolute tolerance parameter passed to the optimization algorithm.
- **epsrel** (*float, optional*) – Relative tolerance parameter passed to the optimization algorithm.

Returns
gcp – The gamma conjugate prior with optimized parameters.

Return type
GammaConjugatePrior

posterior_predictive(*q*, *inplace=False*)

Evaluate the posterior predictive distribution for given heat flow data set $\{q_i\}$.

Parameters

- **q** (*array_like*) – Set of heat flow values.
- **inplace** (*bool, optional*) – If `True`, overwrite the input array. Works only if the input is a `numpy.ndarray` instance.

Returns
pdf – The evaluated posterior predictive PDF of heat flow.

Return type
array_like

Notes

The prior is agnostic to the physical unit of the heat flow values. However, to remain consistent, the posterior predictive and all successive Bayesian updates have to be performed with the same heat flow unit.

posterior_predictive_cdf(*q*, *inplace=False*)

Evaluate the posterior predictive distribution for given heat flow data set $\{q_i\}$.

Parameters

- **q** (*array_like*) – Set of heat flow values.
- **inplace** (*bool, optional*) – If `True`, overwrite the input array. Works only if the input is a `numpy.ndarray` instance.

Returns
cdf – The evaluated posterior predictive CDF of heat flow.

Return type
array_like

Notes

The prior is agnostic to the physical unit of the heat flow values. However, to remain consistent, the posterior predictive and all successive Bayesian updates have to be performed with the same heat flow unit.

probability(*a*, *b*)

Evaluate the probability at parameter points.

Parameters

- **a** (*array_like*) – Set of gamma distribution shape parameters a. Has to be of the same shape as `b`.

- **b** (*array_like*) – Set of gamma distribution scale parameters b. Has to be of the same shape as a.

Returns
p – The evaluated prior probability of the parameter pairs $\{(\alpha_i, \beta_i)\}$.

Return type
array_like

`updated`(*q*)

Perform a Bayesian update given a heat flow data set.

Parameters
q (*array_like*) – Set of heat flow values.

Returns
gcp – An updated prior.

Return type
GammaConjugatePrior

Notes

The prior is agnostic to the physical unit of the heat flow values. However, to remain consistent, all successive updates and the posterior predictive have to be performed with the same heat flow unit.

`visualize`(*ax*, *distributions=None*, *cax=None*, *log_axes=True*, *cmap='inferno'*, *color_scale='log'*, *plot_mean=True*, *q_mean=68.3*, *q_plot=[]*, *qstd_plot=[]*, *n_alpha=101*, *n_beta=100*)

Visualize this GammaConjugatePrior instance on an axis.

Parameters

- **ax** (*matplotlib.axes.Axes*) – The `matplotlib.axes.Axes` to visualize on.
- **distributions** (*Iterable[array_like], optional*) – A set of aggregate heat flow distributions, each given as a one-dimensional `numpy.ndarray` of heat flow values in $\mathrm{mW/m^2}$. Each distribution will be displayed via its α and β maximum likelihood estimate, indicating regions of interest. This may also determine the extent of the plot.
- **cax** (*matplotlib.axes.Axes, optional*) – The `matplotlib.axes.Axes` for plotting a color bar.
- **log_axes** (*bool, optional*) – If `True`, set the axes scale to logarithmic, else use linear axes.
- **cmap** (*str or matplotlib.colors.Colormap, optional*) – Which color map to use for the background probability visualization.
- **color_scale** (*Literal['log','lin'], optional*) – If *'log'*, plot log-probability in background, else plot probability linearly.
- **plot_mean** (*bool, optional*) – If *'False'*, do not plot the mean heat flow lines.
- **q_mean** (*float, optional*) – The global mean heat flow in $\mathrm{mW/m^2}$. The default value is 68.3 from Lucazeau (2019).
- **q_plot** (*Iterable[Tuple[float,float,float,str] | float], optional*) – A set of additional average heat flow values to display. For each q a line through the (α, β) parameter space, enumerating parameter combinations whose distributions average to the given q. Each entry in *q_plot* needs to be either a float *q* or a tuple *(q,amin,amax,c)*, where

amin and *amax* denote the α-interval within which the line should be plotted, and c is the color.

- **qstd_plot** (*Iterable[Tuple[float,float,float,str] | float], optional*) – A set of additional heat flow standard deviations to display. For each *qstd* a line through the (α, β) parameter space, enumerating parameter combinations whose distributions are quantified by a standard deviation *qstd*. Each entry in *qstd_plot* needs to be either a float *qstd* or a tuple *(q,amin,amax,c)*, where *amin* and *amax* denote the α-interval within which the line should be plotted, and c is the color.
- **n_alpha** (*int, optional*) – The number of grid points in the α grid axis.
- **n_beta** (*int, optional*) – The number of grid points in the β grid axis.

Notes

Lucazeau, F. (2019). Analysis and mapping of an updated terrestrial
heat flow data set. Geochemistry, Geophysics, Geosystems, 20, 4001– 4024. https://doi.org/10.1029/2019GC008389

`class` **`HeatFlowPredictive`**(*q*, *x*, *y*, *gcp*, *dmin=20000.0*, *n_bootstrap=1000*)

Posterior predictive distribution of regional heat flow taking into account spatial constraints (i.e. minimum distance) of the heat flow values.

Parameters

- **q** (*array_like*) – Regional distribution of N heat flow values. Has to have the unit that the gamma conjugate prior `gcp` is optimized for.
- **x** (*array_like*) – x coordinates of the heat flow values.
- **y** (*array_like*) – y coordinates of the heat flow values.
- **gcp** (`reheatfunq.regional.GammaConjugatePrior`) – Gamma conjugate prior.
- **dmin** (*float, optional*) – Minimum distance to be enforced between heat flow values of one independent sample (in m).
- **n_bootstrap** (*int, optional*) – Number of randomized selections of `q` subsets adhering to the d_{min} criterion.

`cdf`(*q*, *epsabs=0.0*, *epsrel=1e-10*)

Computes the cumulative distribution function.

Parameters

- **q** (*array_like*) – Heat flow q at which to evaluate the CDF.
- **epsabs** (*float, optional*) – Absolute tolerance parameter passed to the quadrature engines.
- **epsrel** (*float, optional*) – Relative tolerance parameter passed to the quadrature engines.

Returns

cdf – Posterior predictive cumulative distribution of regional heat flow.

Return type
array_like

`pdf`(*q*, *epsabs=0.0*, *epsrel=1e-10*)

Computes the probability distribution function.

Parameters

- **q** (*array_like*) – Heat flow *q* at which to evaluate the CDF.
- **epsabs** (*float, optional*) – Absolute tolerance parameter passed to the quadrature engines.
- **epsrel** (*float, optional*) – Relative tolerance parameter passed to the quadrature engines.

Returns
pdf – Posterior predictive probability distribution of regional heat flow.

Return type
array_like

`default_prior`()

The default gamma conjugate prior from the REHEATFUNQ model description paper (Ziebarth *et al.*, 2022a).

Ziebarth, M. J. and von Specht, S.: REHEATFUNQ 1.4.0:
A model for regional aggregate heat flow distributions and anomaly quantification, EGUsphere [preprint], https://doi.org/10.5194/egusphere-2023-222, 2023.

Notes

This prior is designed for heat flow data in mW/m^2.

CHAPTER

FOUR

ANOMALY QUANTIFICATION

4.1 reheatfunq.anomaly

The *reheatfunq.anomaly* module contains functionality to analyze the strength of heat flow anomalies using the *GammaConjugatePrior* model of regional aggregate heat flow distributions. The module contains the workhorse *HeatFlowAnomalyPosterior* for Bayesian heat flow anomaly strength quantification and *AnomalyLS1980* class to model a fault-generated conductive heat flow anomaly [LS1980]. The class *AnomalyNearestNeighbor* can be used to include the results of an external heat flow anomaly modeling (finite elements etc.) in the REHEATFUNQ analysis. The workflow for anomaly quantification using REHEATFUNQ consists of the following steps:

1. Define the $d_{\min}$ (e.g. $d_{\min} = 20\,\mathrm{km}$)
2. Define the conjugate prior to use. Obtain a *GammaConjugatePrior* instance (e.g. using the REHEATFUNQ default from *default_prior()*).
3. Model the fault-generated heat flow anomaly. So far, the *AnomalyLS1980* and *AnomalyNearestNeighbor* are available for this purpose.
4. Compute the marginal posterior in P_H using the *HeatFlowAnomalyPosterior* class, which takes into consideration the bootstrapped updating of the gamma conjugate prior over the set of $d_{\min}$-conforming subsets of the heat flow data.

4.1.1 Vertical Strike-Slip Fault

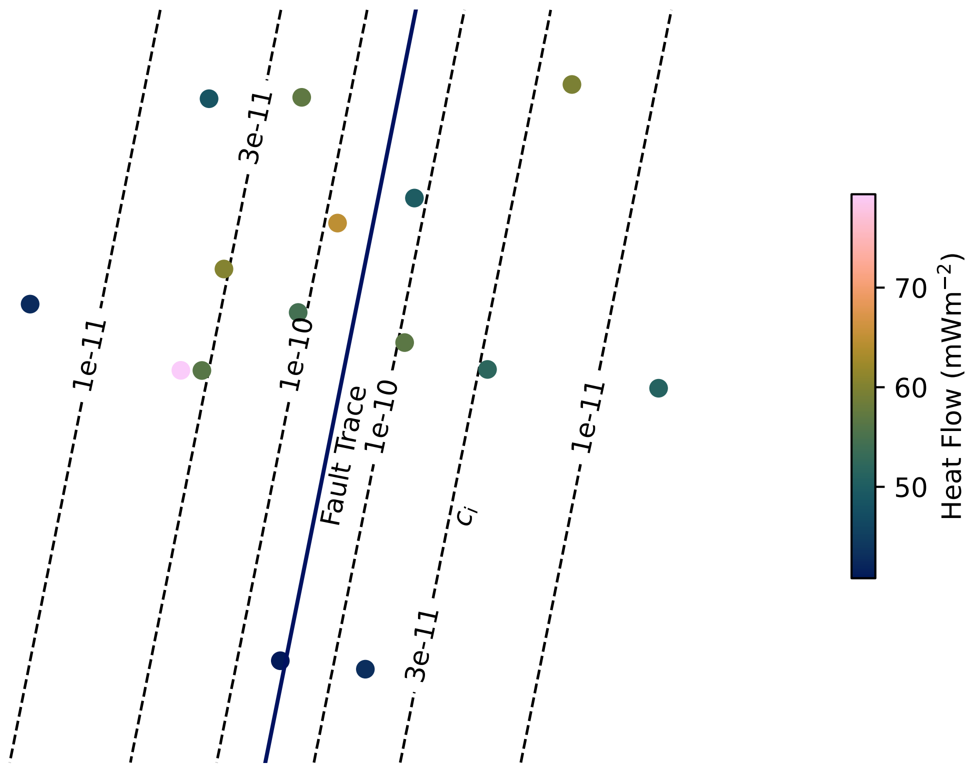
Exemplarily, the following code summarizes the analysis using a heat flow anomaly for a vertical strike-slip fault [LS1980]. First, we generate some toy heat flow data following a gamma distribution. We use the same heat flow data as in the *reheatfunq.regional* example:

```python
import numpy as np
from reheatfunq.anomaly import AnomalyLS1980
rng = np.random.default_rng(123920)
alpha = 53.3
qu = rng.gamma(alpha, size=15)
x = 100e3 * (rng.random(15) - 0.5)
y = 100e3 * (rng.random(15) - 0.5)
```

Generate an obliquely striking vertical strike slip fault and the corresponding conductive heat flow anomaly for a linearly increasing heat production with depth [LS1980]:

```python
fault_trace = np.array([(-20e3, -100e3), (20e3, 100e3)])
anomaly = AnomalyLS1980(fault_trace, 14e3)
xy = np.stack((x,y), axis=1)
```

The data and anomaly look like this (dashed black lines indicate the contours of the heat flow anomaly $c_i = \Delta q_i / P_H$ and the blue line shows the fault trace):



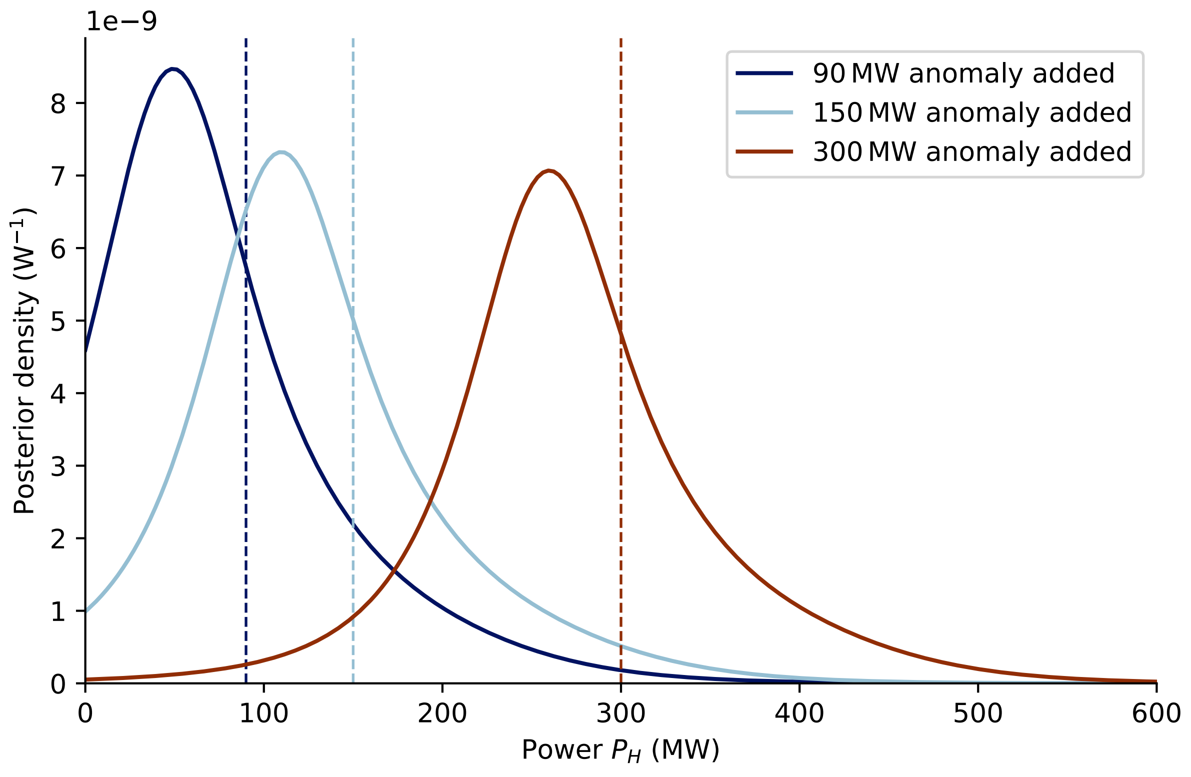
Now compute three sets of heat flow data superposed by heat flow anomalies of 90 MW, 150 MW and 300 MW power:

```
dq = anomaly(xy)
q1 = qu +  90e6 * dq * 1e3
q2 = qu + 150e6 * dq * 1e3
q3 = qu + 300e6 * dq * 1e3
```

Now compute the marginalized posterior distribution of the heat-generating power P_H for the data superposed with the three anomalies:

```
from reheatfunq.anomaly import HeatFlowAnomalyPosterior
from reheatfunq import default_prior
gcp = default_prior()
post1 = HeatFlowAnomalyPosterior(q1, x, y, anomaly, gcp)
post2 = HeatFlowAnomalyPosterior(q2, x, y, anomaly, gcp)
post3 = HeatFlowAnomalyPosterior(q3, x, y, anomaly, gcp)

P_H = np.linspace(0, 600e6, 200)
pdf1 = post1.pdf(P_H)
pdf2 = post2.pdf(P_H)
pdf3 = post3.pdf(P_H)
```



The vertical dashed lines indicate the true anomaly powers.

A detailed use of the anomaly quantification can be found in the Jupyter notebook jupyter/REHEATFUNQ/06-Heat-Flow-Analysis.ipynb.

4.1.2 Custom Heat Flow Anomaly

The next example shows how to use REHEATFUNQ with a heat flow anomaly that has been computed using external code. This is done using the *AnomalyNearestNeighbor* class, and the heat flow anomaly should be specified in terms of the factors c_i (i.e. $\Delta q_i/P_H$). For this purpose, we generate a Gauss-shaped heat flow anomaly that leads to an additional heat flow of $68.3\,\mathrm{mW\,m^{-2}}$ at its center when fed by $10\,\mathrm{MW}$ of heat-generating power:

```
def anomaly_ci(x,y):
    return 68.3e-3 / 10e6 * np.exp(-(x**2 + y**2))
```

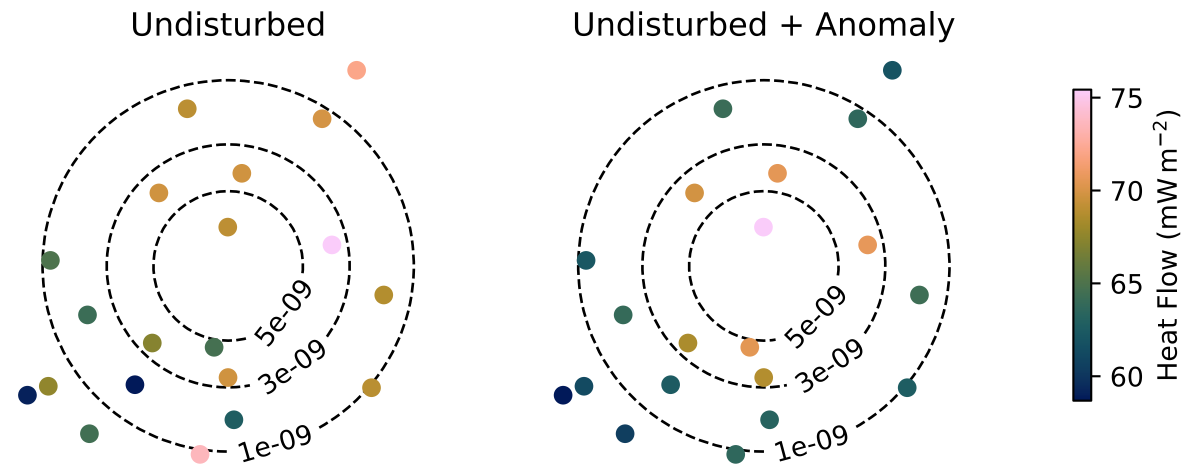
Note here that the c_i should be specified in basic SI units.

We generate some new data superposed by a $10\,\mathrm{MW}$ heat flow anomaly:

```
N = 20
rng = np.random.default_rng(123329773)
xy = 3 * rng.random((N,2)) - 1.5
q0 = 0.39 * rng.gamma(175.2, size=N)
c_i = anomaly_ci(*xy.T)
q = q0 + 1e3 * 10e6 * c_i
```

To adjust the analysis to a custom set of heat flow anomaly factors c_i in $\mathrm{m^{-2}}$, it is sufficient to replace the line `c_i = anomaly_ci(*xy.T)` with whatever code computes or loads the factors. The shape of `c_i` should be compatible to `(N,)`.

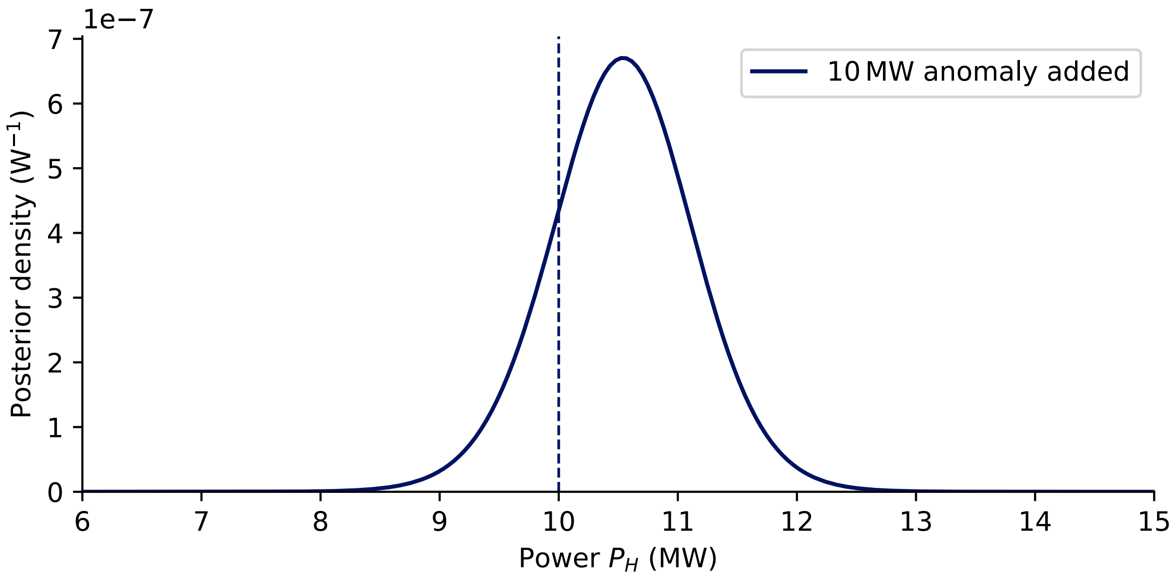
The point set and anomaly generated by the above code should look like this:



With the anomaly factors c_i evaluated at the data locations, we can perform the REHEATFUNQ anomaly analysis using the *AnomalyNearestNeighbor* class:

```
from reheatfunq.anomaly import AnomalyNearestNeighbor
xyc = np.stack((*xy.T, c_i), axis=1)
ann = AnomalyNearestNeighbor(xyc)
hfap = HeatFlowAnomalyPosterior(q, *xy.T, ann, gcp, dmin=0.0)
```

The analysis recovers the anomaly strength:



`class` **`HeatFlowAnomalyPosterior`**(*q*, *x*, *y*, *anomaly*, *gcp*, *dmin=20000.0*, *n_bootstrap='auto'*, *heat_flow_unit='mW/m^2'*, *rng=127*, *rtol=1e-08*, *pdf_algorithm='barycentric_lagrange'*, *bli_max_refinements=7*, *precision='long double'*)

This class evaluates the posterior probability of the strength of a heat flow anomaly, expressed by the frictional power P_H on the fault, using the REHEATFUNQ model of regional heat flow and a set of regional heat flow data.

Parameters

- **q** (*array_like*) – The heat flow data of shape `(N,)`.
- **x** (*array_like*) – The x locations of the heat flow data. Also shape `(N,)`.
- **y** (*array_like*) – The y locations of the heat flow data. Also shape `(N,)`.
- **anomaly** (`reheatfunq.anomaly.anomaly.Anomaly` | *list[*`reheatfunq.anomaly.anomaly.Anomaly`*]* | *list[tuple[float,*`reheatfunq.anomaly.anomaly.Anomaly`*]]*) – The model of the heat flow anomaly that can be evaluated at the data locations. There are three ways to specify this parameter:

  1. A single `Anomaly` instance.
  2. A list of `Anomaly` instances.
  3. A list of tuples `(float,Anomaly)`.

  In the second and third case, the `Anomaly` instances are interpreted as a discretization of the heat transport uncertainty, that is, they should span the space of possible heat transport solution given the knowledge of the problem at hand. The third case lists `float` as prior propabilities for each of the anomaly. This way, one can specify a probability distribution of the heat transport solutions.

- **gcp** (`reheatfunq.regional.GammaConjugatePrior` | *tuple*) – The prior for the regional aggregate heat flow distribution.
- **dmin** (*float, optional*) – The minimum distance between data points (in m). If data points closer than this distance exist in the heat flow data, they are not considered independent and are alternated in the bootstrap.
- **n_bootstrap** (*int, optional*) – The number of permuted heat flow data sets to generate. If no pair of data points is closer than the minimum distance d_min, this parameter has no effect.
- **heat_flow_unit** (*'mW/m^2' | 'W/m^2', optional*) – The unit in which the heat flow data `q` are given.
- **rng** (*int | numpy.random.Generator, optional*) – The random number generator to use or a reproducible seed.
- **rtol** (*float, optional*) – The relative tolerance to aim for in various places within the underlying numerics (quadrature and interpolation).
- **pdf_algorithm** (*"explicit" | "barycentric_lagrange" | "adaptive_simpson", optional*) – The algorithm to use for evaluating the PDF.

  1. `"explicit"`: Explicitly calculate all requested points.
  2. `"barycentric_lagrange"`: First create a barycentric Lagrange interpolator of the PDF which is thereafter used to evaluate the PDF
  3. `"adaptive_simpson"`: Create the adaptive Simpson's rule integrator using the explicitly evaluated PDF. Then use the integrator's polynomials to evaluate the PDF thereafter.

The chosen PDF evaluation algorithm will also be used when evaluating the cumulative distribution functions.

- **bli_max_refinements** (*int, optional*) – The maximum number of refinements of the barycentric Lagrange interpolator. The refinement will stop before if the precision goal is reached. The number of Chebyshev sampling points of the explicitly evaluated PDF grows base-2 exponentially with `bli_max_refinements`.
- **precision** (*'double' | 'long double', optional*) – The precision of the internal numerical computations. The higher the precision, the more likely it is to obtain a precise result for large data sets. The trade off is a longer run time. If the respective flags have been set at compile time, additional options `'float128'` (GCC 128bit floating point), `'dec50'` (boost 50-digit multiprecision), and `'dec100'` (boost 100-digit multiprecision) are available.

cdf(*P_H*)

Evaluate the marginal posterior cumulative distribution of heat-generating power P_H.

Parameters

P_H (*array_like*) – The powers (in W) at which to evaluate the marginal posterior cumulative distribution.

Returns

cdf – The marginal posterior cumulative distribution of heat-generating power P_H evaluated at the given `P_H`.

Return type

numpy.typing.NDArray[numpy.float64]

pdf(*P_H*)

Evaluate the marginal posterior distribution in heat-generating power P_H.

Parameters

P_H (*array_like*) – The powers (in W) at which to evaluate the marginal posterior density.

Returns

pdf – The marginal posterior probability density of heat-generating power P_H evaluated at the given `P_H`.

Return type

numpy.typing.NDArray[numpy.float64]

tail(*P_H*)

Evaluate the posterior tail distribution (complementary cumulative distribution) of heat-generating power P_H.

Parameters

P_H (*array_like*) – The powers (in W) at which to evaluate the marginal posterior tail distribution.

Returns

tail – The marginal posterior tail distribution of heat-generating power P_H evaluated at the given `P_H`.

Return type

numpy.typing.NDArray[numpy.float64]

tail_quantiles(*quantiles*)

Compute posterior tail quantiles, that is, heat-generating powers P_H at which the complementary cumulative distribution of P_H has fallen to level x.

Parameters
`quantiles` (*array_like*) – The tail quantiles to compute.

Returns
P_H – The heat-generating power P_H at which the posterior tail distribution evaluates to `x`.

Return type
numpy.typing.NDArray[numpy.float64]

class Anomaly

Base class for all heat flow anomalies. This base class is useless on its own, it only provides the call signature for the underlying C++ implementation of the anomaly evaluation. The inheriting classes provide the C++ implementation and inherit the `__call__` functionality.

Inheriting classes:

- *AnomalyLS1980*.

`__call__`(*xy*, *P_H=1.0*)

Evaluate the heat flow anomaly at a set of points for a given heat-generating power `P_H`.

Parameters

- **xy** (*double[:,:]*) – Locations at which to evaluate the heat flow anomaly.
- **P_H** (*float*) – The heat-generating power (in W).

Returns
The anomalous heat flow evaluated at the locations, $\{\Delta q_i\} = \{c_i P_H\}$.

Return type
`numpy.ndarray`

class AnomalyLS1980(*const double[:, ::1] xy*, *double d*)

Bases: *Anomaly*

A conductive heat flow anomaly generated by a vertical strike-slip fault whose heat production increases linearly with depth.

This model uses equation (A23b) of Lachenbruch & Sass [LS1980] which is an analytical solution for a straight, infinitely long vertical strike-slip fault in a homogeneous half space. For each queried point, the closest point on the actual, segmented fault is computed using CGAL. The distance between this fault trace point and the query point is plugged into equation (A23b) (see the REHEATFUNQ paper for further details).

The quality of this model's approximation depends on the curvature of the fault and the homogeneity of heat conduction in the crustal volume of interest.

Parameters

- **xy** (*array_like*) – Array of consecutive fault trace coordinates of shape `(N,2)`. The second dimension iterates the coordinate tuple `(x[i], y[i])`.
- **d** (*float*) – Depth of the fault (in m).

`__call__`(*xy*, *P_H=1.0*)

See `Anomaly`.

`length`(*self*) $\rightarrow$ double

`class AnomalyNearestNeighbor`(*const double[:, ::1] xyc*)

Bases: `Anomaly`

A multi-purpose heat flow anomaly class with user-provided anomaly coefficients c_i using nearest neighbor interpolation to obtain the coefficients at the data locations.

This class can be used to provide arbitrary heat flow solutions (e.g. from numerical methods) to REHEATFUNQ. To do so, the coefficients should be provided at the locations of the data later analyzed.

Parameters

`xyc` (*array_like*) – Array of point solutions (x_i, y_i, c_i).

`__call__`(*xy*, *P_H=1.0*)

See `Anomaly`.

The use of this class is demonstrated in the quickstart Jupyter notebook jupyter/Custom-Anomaly.ipynb.

CHAPTER

FIVE

RGRDC

5.1 reheatfunq.coverings

Facilities to compute Random Regional *R*-Disk Coverings (RGRDCs). A RGRDC is a derived product of a global point data set (e.g. a global heat flow database). The covering consists of sequentially generated disks of a radius R randomly distributed over Earth under the constraint that

1. From the set of points within the disk, no two points are closer than the minimum distance d_{min} from each other.
2. No data point is part of a previous disk.
3. The disk center is not contained within an optional exclusion polygon that represents a region of interest for local analysis.
4. There are more than a minimum number of points remaining in the disk.

The function `random_global_R_disk_coverings()` computes RGRDCs. It operates by iteratively drawing random disk centers on the sphere and testing whether all conditions can be met. After a maximum number of disk centers have been drawn, the algorithm terminates. The function is used in the following notebooks:

- jupyter/REHEATFUNQ/03-Gamma-Conjugate-Prior-Parameters.ipynb
- jupyter/REHEATFUNQ/A2-Goodness-of-Fit_R_and-Mixture-Distributions.ipynb
- jupyter/REHEATFUNQ/A6-Comparison-With-Other-Distributions.ipynb
- jupyter/REHEATFUNQ/A5-Uniform-Point-Density.ipynb

The function `conforming_data_selection()` can ensure the d_{min} criterion within a set of heat flow measurements. It proceeds to resolve conflicts to this criterion by iteratively dropping a random data point of a violating data point pair until no more data point pairs violate the criterion.

The function `bootstrap_data_selection()` creates a number of such conforming data selections using random decisions for each conflict.

random_global_R_disk_coverings(*R*, *min_points*, *hf*, *buffered_poly_xy*, *proj_str*, *N=10000*, *MAX_DRAW=100000*, *dmin=0.0*, *seed=982981*, *used_points=None*, *a=6378137.0*)

Uses rejection sampling to draw a number of exclusive regional distributions.

Parameters

- **R** (*float*) – Radius R of the RGRDC (in m).

- **`min_points`** (*`int`*) – Minimum number of points within a distribution after all other conditions are met. If the number of data points is less, the proposed disk is rejected.
- **`hf`** (*`array_like`*) – Array of heat flow data points of shape `(N,3)`, where `N` is the number of data points. The second dimension must contain a tuple (q_i, λ_i, ϕ_i) for each data point, where q_i is the heat flow, λ_i the longitude in degrees, and ϕ_i the latitude in degrees.
- **`buffered_poly_xy`** (*`list[array_like]`*) – List of polygons which will reject disks if their centers fall within one of the polygons. Each element of the list must be a `(M[i],2)`-shaped numpy array where $M[i]$ is the number of points composing the *i* th polygon and the second dimension iterates the coordinates x and y. The coordinates are interpreted within the coordinate system described by the `proj_str` parameter.
- **`proj_str`** (*`str`*) – A PROJ string describing a projected coordinate system within which the polygons supplied in the `buffered_poly_xy` parameter are interpreted.
- **`N`** (*`int, optional`*) – Target number of accepted disks. Might not be reached but can lead to an early exit. The default is high enough that likely `MAX_DRAW` is saturated before.
- **`MAX_DRAW`** (*`int, optional`*) – Maximum number of disk centers to generate. Might not be reached if `N` is small.
- **`dmin`** (*`float, optional`*) – Minimum inter-point distance for the conforming selection criterion (in m).
- **`seed`** (*`int, optional`*) – Seed passed to `np.random.default_rng()`.
- **`used_points`** (*`list[int], optional`*) – A list of data point indices that can be marked as used *a priori*.
- **`a`** (*`float, optional`*) – Large half axis of the sphere used. This parameter is used for a `scipy.spatial.KDTree`-based fast data point query before computing geodesic distances between data points.

Returns

- **valid_points** (*list*) – A list of v centroids of the accepted disks.
- **used_points** (*set*) – A set of all points which are part of an accepted heat flow distribution.
- **distributions** (*list*) – The list of v distributions, each a one-dimensional numpy array of sorted heat flow values.
- **lolas** (*list*) – The list of data point coordinates corresponding to the heat flow data within `distributions`. Each is a two-dimensional numpy array in which the second dimension iterates a tuple (λ, ϕ) of geographic coordinates.
- **distribution_indices** (*list*) – The list of index lists of the data points used in the `distributions`. Each is a one-dimensional array of integer indices into the input data set that compose the corresponding entry of `distributions`. The indices `distribution_indices[i]` are generally not in the same order as the heat flow values in `distributions[i]`.

`conforming_data_selection`(*const double[:, :] xy*, *double dmin_m*, *rng=128*)

This methods applies the spatial data filtering technique described in the paper, sub-sampling the data so that the minimum distance remains above *dmin_m*.

The selection process for non-conforming data pairs is stochastic but reproducible with identical random number generator *rng*.

Parameters

- **xy** (*array_like*) – `(N,2)` array of data points in a projected Euclidean coordinate system (in m).
- **dmin_m** (*float*) – Minimum inter-point distance for the conforming selection criterion (in m).
- **rng** (*int | numpy.random.Generator*) – A seed or random generator to draw from for reproducibility.

Returns
mask – A mask filtering out non-conforming data points.

Return type
numpy.ndarray

`bootstrap_data_selection`(*const double[:, ::1] xy*, *double dmin_m*, *size_t B*, *rng=127*)

Computes a set of bootstrap samples of heat flow data points conforming to the data selection criterion.

Parameters

- **xy** (*array_like*) – `(N,2)` array of data points in a projected Euclidean coordinate system (in m).
- **dmin_m** (*float*) – Minimum inter-point distance for the conforming selection criterion (in m).
- **B** (*int*) – Number of bootstrap samples to draw.
- **rng** (*int | numpy.random.Generator*) – A seed or random generator to draw from for reproducibility.

Returns
subselections – A list of index arrays. Each index array lists the indices of a conforming data selection within the data array `xy`. The number of index arrays is at most `B`. Duplicate data selections are returned only once.

Return type
list

CHAPTER

SIX

HEAT FLOW DATA

6.1 reheatfunq.data

The *reheatfunq.data* module contains a function to load data from the New Global Heat Flow (NGHF) data set of Lucazeau [L2019]. The NGHF data set can be downloaded from the paper's supporting information S02. The function *read_nghf()* can be used as follows:

```python
from reheatfunq.data import read_nghf

nghf_file = 'path/to/NGHF.csv'

nghf_lon, nghf_lat, nghf_hf, nghf_quality, nghf_yr, \
nghf_type, nghf_max_depth, nghf_uncertainty, indexmap \
   = read_nghf(nghf_file)
```

The Jupyter notebook jupyter/REHEATFUNQ/01-Load-and-filter-NGHF.ipynb illustrates how this function was used in the derivation of the REHEATFUNQ model.

read_nghf(*f*)

This function reads the NGHF data base.

Parameters

f (*str | pathlib.Path*) – The file path to the `NGHF.csv` file of Lucazeau [L2019].

Returns

- **nghf_lon** (*list*) – Longitudes of the data points. This field is mandatory.
- **nghf_lat** (*list*) – Latitudes of the data points. This field is mandatory.
- **nghf_hf** (*list*) – Heat flow of the data points in $\mathrm{mW m^{-2}}$. This field is mandatory.
- **nghf_quality** (*list*) – Quality of the data points. If defined, one of 'A', 'B', 'C', 'D' or 'Z'.
- **nghf_yr** (*list*) – Measurement year of the data points. This field is mandatory.
- **nghf_type** (*list*) – Whether the heat flow data point is continental or oceanic. If defined, one of 'land' or 'ocean'. This field is mandatory.
- **nghf_max_depth** (*list*) – Maximum depth used in estimating the temperature gradient. If the field is empty, returns -9999999.

- **nghf_uncertainty** (*list*) – Relative uncertainty of the data points. Returns 0.1 for 'A' quality, 0.2 for 'B' quality, 0.3 for 'C' quality, and infinity otherwise.
- **indexmap** (*dict*) – Maps indices within the returned arrays to lines in the `NGHF.csv` table. If `i` was an index within the returned arrays, then `j = indexmap[i]` is the row within the NGHF table.

CHAPTER

SEVEN

RESILIENCE ANALYSIS

7.1 reheatfunq.resilience

This module contains functions to evaluate the performance of the REHEATFUNQ model for artifical gamma-distributed data, and its resilience against non-gamma regional aggregate heat flow distributions.

The function *test_performance_cython()* can be used to investigate how the REHEATFUNQ model performs for data drawn from a gamma distribution, distributed randomly within a $R = 80\,\mathrm{km}$ disk, and superposed by an *AnomalyLS1980* anomaly. The sample size, the gamma distribution parameters, and the prior parameters are the tweakable parameters of this function. It is used in the Jupyter notebook jupyter/REHEATFUNQ/A3-Posterior-Impact.ipynb.

The function *test_performance_mixture_cython()* can be used to investigate how well the REHEATFUNQ model performs if data is not drawn from a gamma distribution but from a two-component Gaussian mixture distribution. That is, it is a resilience test that can be tweaked to a certain class of regional aggregate heat flow distributions. It is also used in the Jupyter notebook jupyter/REHEATFUNQ/A3-Posterior-Impact.ipynb.

The functions *generate_synthetic_heat_flow_coverings_mix2()* and *generate_synthetic_heat_flow_coverings_mix3(* generate synthetic RGRDCs that can mimic RGRDCs from real data. The two functions proceed as follows:

1. Input the structure of the real-world data RGRDC: Represent each disk by a tuple (N, k, θ), where N is the sample size and (k, θ) is the maximum likelihood estimate of the gamma distribution for the regional aggregate heat flow distribution associated to the disk.
2. Define a two-component (`_mix2`) or three-component (`_mix3`) "Gaussian" mixture distribution that describes the relative error distribution of the heat flow data. ("Gaussian" because we ignore the negative real line)
3. For a number of `M` times, repeat the following steps to generate one synthetic RGRDC:
    - for each (N, k, θ), draw a sample from the (k, θ) gamma distribution
    - draw a random relative error from the "Gaussian" mixture distribution and superpose the relative error randomly in positive or negative direction
    - accept or reject according to filter criteria (heat flow positivity and max heat flow)
    - repeat until N heat flow values are found

The two functions are used in the Jupyter notebook jupyter/REHEATFUNQ/A2-Goodness-of-Fit_R_and-Mixture-Distributions.ipynb.

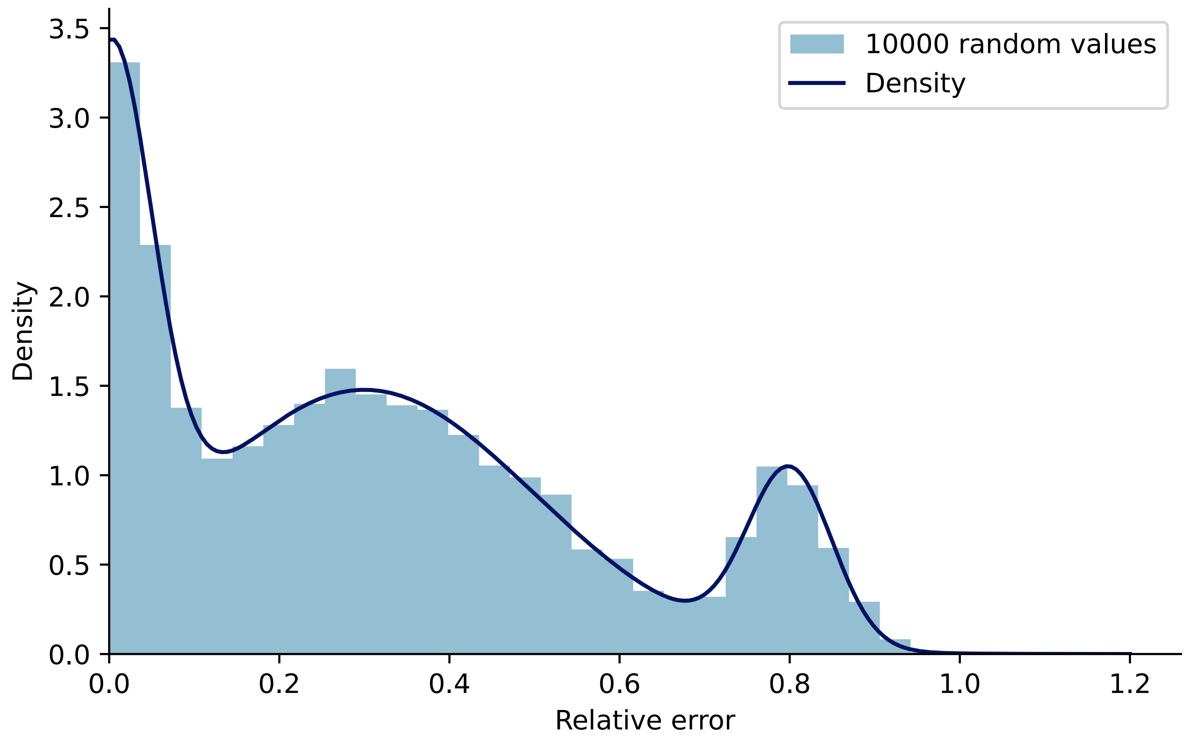
The function *reheatfunq.resilience.generate_normal_mixture_errors_3()* is an interface to the generation of the three-component "Gaussian" mixture distribution described above. An example for the distribution can be generated from this code:

```python
from reheatfunq.resilience import \
    generate_normal_mixture_errors_3
```

(continues on next page)

(continued from previous page)

```
X00 = 0.0
X01 = 0.30
X02 = 0.8
W0 = 0.3
S0 = 0.05
S1 = 0.2
W1 = 0.6
S2 = 0.05
X = generate_normal_mixture_errors_3(10000, W0, X00, S0, W1,
                                     X01, S1, X02, S2, 2089)
```



It is used in the Jupyter notebook jupyter/REHEATFUNQ/A2-Goodness-of-Fit_R_and-Mixture-Distributions.ipynb.

test_performance_cython(*long[:] Nset*, *size_t M*, *double P_MW*, *double K*, *double T*, *double[:] quantile*, *double PRIOR_P*, *double PRIOR_S*, *double PRIOR_N*, *double PRIOR_V*, *double amin=1.0*, *short verbose=True*, *short show_failures=False*, *size_t seed=848782*, *short use_cpp_quantiles=True*, *double tol=1e-3*, *unsigned char nthread=0*)

Tests the performance of the gamma model (with and without prior) for synthetic data sets that do not stem from a gamma distribution. The analysis is performed for synthetic data randomly distributed within an $80\,\mathrm{km}$ radius disk with a straight-line fault passing through its center.

Parameters

- **Nset** (*array_like*) – Sample sizes $\{N_i\}$ for which to perform the test.

- **M** (*int*) – Number of repetition per sample size.
- **P_MW** (*float*) – Power of the anomaly.
- **K** (*float*) – Gamma distribution shape parameter k.
- **T** (*float*) – Gamma distribution scale parameter θ.
- **quantile** (*array_like*) – Array of anomaly P_H posterior quantiles to evaluate. The array must be either 4 or 41 elements in size.
- **PRIOR_P** (*float*) – Parameter p of the gamma conjugate prior.
- **PRIOR_s** (*float*) – Parameter s of the gamma conjugate prior.
- **PRIOR_N** (*float*) – Parameter n of the gamma conjugate prior.
- **PRIOR_V** (*float*) – Parameter ν of the gamma conjugate prior.
- **amin** (*float*) – The minimum shape parameter α of the gamma distribution. Has to be positive.
- **verbose** (*bool, optional*) – If `True`, print some progress information.
- **show_failures** (*bool, optional*) – Currently without effect.
- **seed** (*int, optional*) – Random number generator seed for reproducibliity.
- **use_cpp_quantiles** (*bool, optional*) – Currently without effect.
- **tol** (*float, optional*) – Quantile inversion tolerance passed to the algorithms.

Returns
: **res** – Quantiles of the P_H posteriors. The array has the shape `(2, len(Nset), len(quantile), M)`.

Return type
: numpy.ndarray

test_performance_mixture_cython(*long[:] Nset*, *size_t M*, *double P_MW*, *double x0*, *double s0*, *double a0*, *double x1*, *double s1*, *double a1*, *double[:] quantile*, *double PRIOR_P*, *double PRIOR_S*, *double PRIOR_N*, *double PRIOR_V*, *double amin*, *short verbose=True*, *short show_failures=False*, *size_t seed=848782*, *short use_cpp_quantiles=True*, *double tol=1e-4*)

Tests the performance of the gamma model (with and without prior) for synthetic data sets that do not stem from a gamma distribution. The analysis is performed for synthetic data randomly distributed within an $80\,\mathrm{km}$ radius disk with a straight-line fault passing through its center.

Quantiles are computed both for the prior with the supplied parameters and for the “uninformed” prior ($p = 1$, $s = n = \nu = 0$).

Parameters

- **Nset** (*array_like*) – Sample sizes $\{N_i\}$ for which to perform the test.
- **M** (*int*) – Number of repetition per sample size.
- **P_MW** (*float*) – Power of the anomaly.
- **x0** (*float*) – Location of the first normal mixture component.
- **s0** (*float*) – Standard deviation of the first normal mixture component.
- **a0** (*float*) – Weight of the first normal mixture component.
- **x1** (*float*) – Location of the second normal mixture component.

- **s1** (*float*) – Standard deviation of the second normal mixture component.
- **quantile** (*array_like*) – Array of anomaly P_H posterior quantiles to evaluate. The array must be either 4 or 41 elements in size.
- **PRIOR_P** (*float*) – Parameter p of the gamma conjugate prior.
- **PRIOR_s** (*float*) – Parameter s of the gamma conjugate prior.
- **PRIOR_N** (*float*) – Parameter n of the gamma conjugate prior.
- **PRIOR_V** (*float*) – Parameter ν of the gamma conjugate prior.
- **amin** (*float*) – The minimum shape parameter α of the gamma distribution. Has to be positive.
- **verbose** (*bool, optional*) – If `True`, print some progress information.
- **show_failures** (*bool, optional*) – Currently without effect.
- **seed** (*int, optional*) – Random number generator seed for reproduciblity.
- **use_cpp_quantiles** (*bool, optional*) – Currently without effect.
- **tol** (*float, optional*) – Quantile inversion tolerance passed to the algorithms.

Returns

res – Quantiles of the P_H posteriors. The array has the shape `(2, len(Nset), len(quantile), M)`.

Return type

numpy.ndarray

generate_synthetic_heat_flow_coverings_mix2(*const double[:] k*, *const double[:] t*, *const long[:] N*, *long M*, *double hf_max*, *double w0*, *double x00*, *double s0*, *double x10*, *double s1*, *size_t seed*, *unsigned short nthread*)

Generate synthetic heat flow coverings using a two component normal mixture distribution as an error distribution.

Parameters

- **k** (*array_like*) – `M` gamma distribution shape parameters k.
- **t** (*array_like*) – `M` gamma distribution scale parameters θ.
- **N** (*array_like*) – `M` sample sizes to draw from the corresponding gamma distributions.
- **M** (*int*) – Number of RGRDCs to draw.
- **hf_max** (*float*) – Threshold below which to accept heat flow values.
- **w0** (*float*) – Weight of the first normal distribution describing the error mixture distribution.
- **x00** (*float*) – Location of the first normal distribution.
- **s0** (*float*) – Standard deviation of the first normal distribution.
- **x10** (*float*) – Location of the second normal distribution.
- **s1** (*float*) – Standard deviation of the second normal distribution.
- **seed** (*int*) – Seed by which to initialize the random number generation.
- **nthread** (*int*) – Number of threads to use. In combination with seed, this fixes the sequence of random number generation used in this run. Keep both values the same to obtain reproducible results.

Returns
res – List of lists distributions forming the RGRDCs.

Return type
list[list]

`generate_synthetic_heat_flow_coverings_mix3`(*list k*, *list t*, *list N*, *double hf_max*, *double w0*, *double x00*, *double s0*, *double w1*, *double x10*, *double s1*, *double x20*, *double s2*, *size_t seed*, *unsigned short nthread*)

Generate synthetic heat flow coverings using a three component normal mixture distribution as an error distribution.

Parameters

- **k** (*list[array_like]*) – M arrays of gamma distribution shape parameters k.
- **t** (*list*) – M arrays of gamma distribution scale parameters θ.
- **N** (*list*) – M arrays of sample sizes to draw from the corresponding gamma distributions.
- **hf_max** (*float*) – Threshold below which to accept heat flow values.
- **w0** (*float*) – Weight of the first normal distribution describing the error mixture distribution.
- **x00** (*float*) – Location of the first normal distribution.
- **s0** (*float*) – Standard deviation of the first normal distribution.
- **w1** (*float*) – Weight of the second normal distribution describing the error mixture distribution.
- **x10** (*float*) – Location of the second normal distribution.
- **s1** (*float*) – Standard deviation of the second normal distribution.
- **x20** (*float*) – Location of the third normal distribution.
- **s2** (*float*) – Standard deviation of the third normal distribution.
- **seed** (*int*) – Seed by which to initialize the random number generation.
- **nthread** (*int*) – Number of threads to use. In combination with seed, this fixes the sequence of random number generation used in this run. Keep both values the same to obtain reproducible results.

Returns
res – List of lists of distributions forming the RGRDCs.

Return type
list[list]

`generate_normal_mixture_errors_3`(*size_t N*, *double w0*, *double x00*, *double s0*, *double w1*, *double x10*, *double s1*, *double x20*, *double s2*, *size_t seed*)

Draw random numbers from the three-component normal mixture distribution.

Parameters

- **N** (*int*) – Number of random numbers to generate.
- **w0** (*float*) – Weight of the first mixture component.
- **x00** (*float*) – Center of the first mixture component.
- **s0** (*float*) – Standard deviation of the first mixture component.
- **w1** (*float*) – Weight of the second mixture component.

- **x10** (*float*) – Center of the second mixture component.
- **s1** (*float*) – Standard deviation of the second mixture component.
- **x20** (*float*) – Center of the third mixture component.
- **s2** (*float*) – Standard deviation of the third mixture component.
- **seed** (*int*) – Random number generator seed for reproducibility.

Returns
X – Random values.

Return type
numpy.ndarray

CHAPTER

EIGHT

BIBLIOGRAPHY

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[L2019] Lucazeau, F. (2019). Analysis and mapping of an updated terrestrial heat flow data set. Geochemistry, Geophysics, Geosystems, 20, 4001-4024. https://doi.org/10.1029/2019GC008389

[LS1980] Lachenbruch, A. H., and Sass, J. H. (1980), Heat flow and energetics of the San Andreas Fault Zone, J. Geophys. Res., 85(B11), 6185-6222, https://doi.org/10.1029/JB085iB11p06185

[Miller1980] Miller, Robert B. (1980). Bayesian Analysis of the Two-Parameter Gamma Distribution, Technometrics, 22:1, 65-69. https://doi.org/10.1080/00401706.1980.10486102

PYTHON MODULE INDEX

r

reheatfunq.anomaly, 19
reheatfunq.coverings, 27
reheatfunq.data, 31
reheatfunq.regional, 9
reheatfunq.resilience, 33

INDEX

Symbols

`__call__()` (*Anomaly method*), 25
`__call__()` (*AnomalyLS1980 method*), 25
`__call__()` (*AnomalyNearestNeighbor method*), 26

A

`Anomaly` (*class in reheatfunq.anomaly.anomaly*), 25
`AnomalyLS1980` (*class in reheatfunq.anomaly*), 25
`AnomalyNearestNeighbor` (*class in reheatfunq.anomaly*), 26

B

`bootstrap_data_selection()` (*in module reheatfunq.coverings*), 29

C

`cdf()` (*HeatFlowAnomalyPosterior method*), 24
`cdf()` (*HeatFlowPredictive method*), 17
`conforming_data_selection()` (*in module reheatfunq.coverings*), 28

D

`default_prior()` (*in module reheatfunq.regional*), 18

G

`GammaConjugatePrior` (*class in reheatfunq.regional*), 11
`generate_normal_mixture_errors_3()` (*in module reheatfunq.resilience*), 37
`generate_synthetic_heat_flow_coverings_mix2()` (*in module reheatfunq.resilience*), 36
`generate_synthetic_heat_flow_coverings_mix3()` (*in module reheatfunq.resilience*), 37

H

`HeatFlowAnomalyPosterior` (*class in reheatfunq.anomaly*), 22
`HeatFlowPredictive` (*class in reheatfunq.regional*), 17

K

`kullback_leibler()` (*GammaConjugatePrior method*), 11

L

`length()` (*AnomalyLS1980 method*), 26
`log_likelihood()` (*GammaConjugatePrior method*), 12
`log_probability()` (*GammaConjugatePrior method*), 12

M

`maximum_likelihood_estimate()` (*GammaConjugatePrior static method*), 12
`minimum_surprise_estimate()` (*GammaConjugatePrior static method*), 13
`minimum_surprise_estimate_cache()` (*GammaConjugatePrior static method*), 14
`mle()` (*GammaConjugatePrior static method*), 14
`module`
  `reheatfunq.anomaly`, 19
  `reheatfunq.coverings`, 27
  `reheatfunq.data`, 31
  `reheatfunq.regional`, 9
  `reheatfunq.resilience`, 33

P

`pdf()` (*HeatFlowAnomalyPosterior method*), 24
`pdf()` (*HeatFlowPredictive method*), 18
`posterior_predictive()` (*GammaConjugatePrior method*), 15
`posterior_predictive_cdf()` (*GammaConjugatePrior method*), 15
`probability()` (*GammaConjugatePrior method*), 15

R

`random_global_R_disk_coverings()` (*in module reheatfunq.coverings*), 27
`read_nghf()` (*in module reheatfunq.data*), 31
`reheatfunq.anomaly`
  `module`, 19
`reheatfunq.coverings`
  `module`, 27
`reheatfunq.data`
  `module`, 31
`reheatfunq.regional`

module, 9
reheatfunq.resilience
module, 33

T

tail() (*HeatFlowAnomalyPosterior method*), 24
tail_quantiles() (*HeatFlowAnomalyPosterior method*), 24
test_performance_cython() (*in module reheatfunq.resilience*), 34
test_performance_mixture_cython() (*in module reheatfunq.resilience*), 35

U

updated() (*GammaConjugatePrior method*), 16

V

visualize() (*GammaConjugatePrior method*), 16