
hIPPYlib Documentation

Release 3.1.0

Umberto Villa, Noemi Petra, Omar Ghattas

Dec 22, 2022

Contents

1	Installation	3
2	Changelog	7
3	How to Contribute	13
4	hippylib	19
	Python Module Index	59
	Index	61

[illegible]

<https://hippylib.github.io>

For building instructions, see the file `INSTALL.md`. Copyright information and licensing restrictions can be found in the file `COPYRIGHT`.

Conceptually, `hIPPYlib` can be viewed as a toolbox that provides the building blocks for experimenting new ideas and developing scalable algorithms for PDE-constrained deterministic and Bayesian inverse problems.

Noise and prior covariance operators are modeled as inverses of elliptic differential operators allowing us to build on existing fast multigrid solvers for elliptic operators without explicitly constructing the dense covariance operator.

`hiPPYlib` provides a robust implementation of the inexact Newton-conjugate gradient algorithm to compute the maximum a posteriori (MAP) point. The gradient and Hessian actions are computed via their weak form specification in `FEniCS` by constraining the state and adjoint variables to satisfy the forward and adjoint problem. The Newton system is solved inexactly by early termination of CG iterations via Eisenstat-Walker (to prevent oversolving) and Steihaug (to avoid negative curvature) criteria. Two globalization techniques are available to the user: Armijo backtracking line search and trust region.

In `hippylib`, the posterior covariance is approximated by the inverse of the Hessian of the negative log posterior evaluated at the MAP point. This Gaussian approximation is exact when the parameter-to-observable map is linear; otherwise, its logarithm agrees to two derivatives with the log posterior at the MAP point, and thus it can serve as a

proposal for Hessian-based Markov chain Monte Carlo (MCMC) methods. `hIPPYlib` makes the construction of the posterior covariance tractable by invoking a low-rank approximation of the Hessian of the log likelihood.

`hIPPYlib` also offers scalable methods for sample generation. To sample large scale spatially correlated Gaussian random fields from the prior distribution, `hIPPYlib` implements a new method that strongly relies on the structure of the covariance operator defined as the inverse of a differential operator: by exploiting the assembly procedure of finite element matrices `hIPPYlib` constructs a sparse Cholesky-like rectangular decomposition of the precision operator. To sample from a local Gaussian approximation to the posterior (such as at the MAP point) `hIPPYlib` exploits the low rank factorization of the Hessian of the log likelihood to correct samples from the prior distribution. Finally, to explore the posterior distribution, `hIPPYlib` implements dimension independent MCMC sampling methods enchanted by Hessian information.

Finally, randomized and probing algorithms are available to compute the pointwise variance of the prior/posterior distribution and the trace of the covariance operator.

Installation

Inverse Problem PYthon library

[illegible]

`https://hippylib.github.io`

hIPPYlib depends on FEniCS version 2019.1.

FEniCS needs to be built with the following dependencies enabled:

- `numpy`, `scipy`, `matplotlib`, `mpi4py`
- `PETSc` and `petsc4py` (version 3.10.0 or above)
- `SLEPc` and `slepc4py` (version 3.10.0 or above)
- **PETSc dependencies:** `parmetis`, `scotch`, `suitesparse`, `superlu_dist`, `ml`, `hypre`
- (optional): `mshr`, `jupyter`

1.1 Install hIPPYlib using pip

1.1.1 Latest release

With the supported version of FEniCS and its dependencies installed on your machine, hIPPYlib can be installed via pip as follows

```
pip install hippylib --user
```

In order for pip to install extra requirements (e.g. Jupyter) the following command should be used

```
pip install hippylib[notebook] --user
```

1.1.2 Development version/topic branches

To pip install the development version of hIPPYlib use the command

```
pip install -e git+https://github.com/hippylib/hippylib.git@master#egg=hippylib
```

To pip install a topic branch (e.g. the 2019.1-dev2 branch) use

```
pip install -e git+https://github.com/hippylib/hippylib.git@2019.1-dev2#egg=hippylib
```

NOTE: hIPPYlib applications and tutorials can also be executed directly from the source folder without pip installation.

1.2 Build the hIPPYlib documentation using Sphinx

To build the documentation you need to have sphinx (tested on v.1.7.5), m2r and sphinx_rtd_theme - all of these can be installed via pip.

To build simply run `make html` from doc folder.

1.3 FEniCS installation

1.3.1 Install FEniCS from Conda (Linux or MacOS)

To use the prebuilt Anaconda Python packages (Linux and Mac only), first install [Anaconda3](#), then run following commands in your terminal:

```
conda create -n fenics-2019.1 -c conda-forge fenics==2019.1.0 matplotlib scipy jupyter
```

Note: FEniCS Anaconda recipes are maintained by the FEniCS community and distributed binary packages do not have a full feature set yet, especially regarding sparse direct solvers and input/output facilities.

1.3.2 Run FEniCS from Docker (Linux, MacOS, Windows)

An easy way to run FEniCS is to use their prebuilt Docker images.

First you will need to install [Docker](#) on your system. MacOS and Windows users should preferably use Docker for Mac or Docker for Windows — if it is compatible with their system — instead of the legacy version Docker Toolbox.

Among the many docker's workflow discussed [here](#), we suggest using the `Jupyter notebook`.

Docker for Mac, Docker for Windows and Linux users (Setup and first use instructions)

We first create a new Docker container to run the `jupyter-notebook` command and to expose port 8888. In a command line shell type:

```
docker run --name hippylib-nb -w /home/fenics/hippylib -v $(pwd):/home/fenics/
↪hippylib \
    -d -p 127.0.0.1:8888:8888 hippylib/fenics \
    'jupyter-notebook --ip=0.0.0.0'
docker logs hippylib-nb
```

The notebook will be available at `http://localhost:8888/?token=<security_token_for_first_time_connection>` in your web browser. From there you can run the interactive notebooks or create a new shell (directly from your browser) to run python scripts.

Docker Toolbox users on Mac/Windows (Setup and first use instructions)

Docker Toolbox is for older Mac and Windows systems that do not meet the requirements of Docker for Mac or Docker for Windows. Docker Toolbox will first create a lightweight linux virtual machine on your system and run docker from the virtual machine. This has implications on the workflow presented above.

We first create a new Docker container to run the `jupyter-notebook` command and to expose port 8888 on the virtual machine. In a command line shell type:

```
docker run --name hippylib-nb -w /home/fenics/hippylib -v $(pwd):/home/fenics/
↪hippylib \
    -d -p $(docker-machine ip $(docker-machine active)):8888:8888 \
    hippylib/fenics 'jupyter-notebook --ip=0.0.0.0'
docker logs hippylib-nb
```

To find out the IP of the virtual machine we type:

```
docker-machine ip $(docker-machine active)
```

The notebook will be available at `http://<ip-of-virtual-machine>:8888/?token=<security_token_for_first_time_connection>` in your web browser. From there you can run the interactive notebooks or create a new shell (directly from your browser) to run python scripts.

Subsequent uses

The docker container will continue to run in the background until we stop it:

```
docker stop hippylib-nb
```

To start it again just run:

```
docker start hippylib-nb
```

If you would like to see the log output from the Jupyter notebook server (e.g. if you need the security token) type:

```
docker logs hippylib-nb
```

1.3.3 Other ways to build FEniCS

For instructions on other ways to build FEniCS, we refer to the FEniCS project [download page](#). Note that this instructions always refer to the latest version of FEniCS which may or may not be yet supported by hIPPYlib. Always check the hIPPYlib website for supported FEniCS versions.

[illegible]

2.1 Version 3.1.0, released on Dec 21, 2022

- ## 2.2 Version 3.0.0, released on Feb 2, 2020

- 7

- Modify `PointwiseObservation` so that ordering of observations targets is respected also in parallel. Setting the flag `prune_and_sort` to `True` restores previous behavior.
- Remove unused input `tol` from `Model.solveFwd`, `Model.solveAdj`, `Model.solveFwdIncremental`, `Model.solveAdjIncremental` and from related classes.
- Use `argparse` to set parameters in application drivers from command line
- Use `dl.XDMFFile` to export solutions for visualization in Paraview in all application drivers
- Implement accuracy enhanced SVD algorithm in `randomizedSVD.py`
- Add forward UQ capabilities, using Taylor approximations as control variates
- Introduce hIPPYlib's wrappers to `petsc4py.PETSc.KSP`
- Add reduction operations useful when solving different PDEs concurrently on each process
- Increase coverage of unit testing in CI

2.3 Version 2.3.0, released on Sept 6, 2019

- Reimplement `BiLaplacianPrior` and `MollifiedBiLaplacianPrior` using a more general framework `SqrtPrecisionPDE_Prior`, which also supports Gaussian vector fields.
- Update the prior distribution in `model_subsurf.py` and `tutorial/3_SubsurfaceBayesian.ipynb` to use Robin boundary conditions to alleviate boundary artifacts.
- Update the data misfit term in `model_ad_diff.py` and `tutorial/4_AdvectionDiffusionBayesian.ipynb` to use discrete observations in space and time.

2.4 Version 2.2.1, released on March 28, 2019

- Bug fix missing `mpi_comm` in `TimeDependentVector`
- Bug fix in the initialization of the global variable `parRandom`

2.5 Version 2.2.0, released on Dec 12, 2018

- Add new class `GaussianRealPrior` that implements a finite-dimensional Gaussian prior
- Add a `callback` user-defined function that can be called at the end of each inexact Newton CG iteration
- Add a `__version__` and `version_info` attribute to `hIPPYlib`
- Add `setup.py` to (optionally) install hIPPYlib via `pip`
- Add deprecation mechanism
- Deprecate `TimeDependentVector.copy(other)` in favor of `TimeDependentVector.copy()` for consistency with `dolfin.Vector.copy`
- Deprecate `_BiLaplacianR.inner(x,y)` for consistency with `dolfin.Matrix`
- CI enhancement via build matrix

2.6 Version 2.1.1, released on Oct 23, 2018

- Update `README.md` and `paper` according to JOSS reviewers's comments
- Add contributing guidelines
- Fix some typos in notebooks (thanks to **Christian Boehm**)

2.7 Version 2.1.0, released on July 18, 2018

- Alleviate boundary artifacts (inflation of marginal variance) in Bilaplacian-like priors using Robin boundary conditions
- Allow the user to select different matplotlib colormaps in jupyter notebooks
- Buxfix in the acceptance ratio of the gpCN MCMC proposal

2.8 Version 2.0.0, released on June 15, 2018

- Introduce capabilities for non-Gaussian Bayesian inference using Mark Chain Monte Carlo methods. Kernels: mMALA, pCN, gpCN, IS. **Note: API subject to change**
- Support domain-decomposition parallelization (new parallel random number generator, and new randomized eigensolvers)
- The parameter, usually labeled `a`, throughout the library, has been renamed to `m`, for model parameter. Interface changes:
 - `PDEProblem.eval_da` → `PDEProblem.evalGradientParameter`
 - `Model.applyWua` → `Model.applyWum`
 - `Model.applyWau` → `Model.applyWmu`
 - `Model.applyRaa` → `Model.applyWmm`
 - `gda_tolerance` → `gdm_tolerance` in the parameter list for Newton and QuasiNewton optimizers
 - `gn_approx` → `gass_newton_approx` as parameter in function to compute Hessian/linearization point in classes `Model`, `PDEProblem`, `Misfit`, `Qoi`, `ReducedQoi`
- Organize `hippylib` in subpackages
- Add `sphinx` documentation (thanks to **E. Khattatov** and **I. Ambartsumyan**)

2.9 Version 1.6.0, released on May 16, 2018

- **Bugfix** in `PDEVariationalProblem.solveIncremental` for non self-adjoint models
- Add new estimator for the trace and diagonal of the prior covariance using randomized eigendecomposition
- In all examples and tutorial, use enviromental variable `HIPPYLIB_BASE_DIR` (if defined) to add `hippylib` to `PYTHONPATH`

2.10 Version 1.5.0, released on Jan 24, 2018

- Add support for `FEniCS 2017.2`

2.11 Version 1.4.0, released on Nov 8, 2017

- Add support for `Python 3`
- Enchantments in `PDEVariationalProblem`: it now supports multiple Dirichlet condition and vectorial/mixed function spaces
- Bugfix: Set the correct number of global rows, when targets points fall outside the computational domain
- More extensive testing with Travis Integration

2.12 Version 1.3.0, released on June 28, 2017

- Improve `hashdist` installation support
- Switch license to GPL-2
- Add support for `FEniCS 2017.1`

2.13 Version 1.2.0, released on April 24, 2017

- Update instruction to build `FEniCS: hashdist and docker`
- Update notebook to `nbformat 4`
- Let `FEniCS 2016.2` be the preferred version of `FEniCS`
- Add Travis integration

2.14 Version 1.1.0, released on Nov 28, 2016

- Add partial support for `FEniCS 2016.1` (Applications and Tutorial)
- Improve performance of the randomized eigensolvers

2.15 Version 1.0.2, released on Sep 30, 2016

- Use `vector2Function` to safely convert `dolfin.Vector` to `dolfin.Function`
- Optimize the `PDEVariationalProblem` to exploit the case when the forward problem is linear
- Update notebook `1_FEniCS101.ipynb`

2.16 Version 1.0.1, released on Aug 25, 2016

- Add support in `hippylib.Model` and `hippylib.Misfit` for misfit functional with explicit dependence on the parameter

2.17 Version 1.0.0, released on Aug 8, 2016

- Uploaded to <https://hippylib.github.io>.
- Initial release under GPL-3.

```
Inverse Problem PYthon library
```

```

/  |      /  |      /  |      /  |      /  |      /  |      /  |      /  |
$$ |____  $$$$$$/ $$$$$$$ $$$$$$$$ |$$ /$$/ $$ |$$/ $$ |____
$$      $$ |  $$ |__$$ |$$ |__$$ |  $$ \/$$/ $$ | /  |$$
$$$$$$$ |  $$ |  $$  $$/$$  $$/$$  $$/$$  $$ |$$ |$$$$$$$ |
$$ |  $$ |  $$ |  $$$$$$/ $$$$$$/  $$$$$/  $$ |$$ |$$ |  $$ |
$$ |  $$ |__$$ |__ $$ |  $$ |  $$ |  $$ |  $$ |$$ |$$ |__$$ |
$$ |  $$ |/$$ |$$ |  $$ |  $$ |  $$ |  $$ |$$ |$$  $$/$$
$$/  $$/ $$$$$$/ $$/  $$/  $$/  $$/  $$/  $$/ $$$$$$/

```

```
https://hippylib.github.io
```


CHAPTER 3

How to Contribute

The `hIPPYlib` team welcomes contributions at all levels: bugfixes, code improvements, new capabilities, improved documentation, or new examples/tutorials.

Use a pull request (PR) toward the `hippylib:master` branch to propose your contribution. If you are planning significant code changes, or have any questions, you should also open an [issue](#) before issuing a PR.

See the *Quick Summary* section for the main highlights of our GitHub workflow. For more details, consult the following sections and refer back to them before issuing pull requests:

- *GitHub Workflow*
 - *hIPPYlib Organization*
 - *New Feature Development*
 - *Developer Guidelines*
 - *Pull Requests*
 - *Pull Request Checklist*
- *Automated Testing*
- *Contact Information*

Contributing to `hIPPYlib` requires knowledge of Git and, likely, inverse problems. If you are new to Git, see the [GitHub learning resources](#). To learn more about inverse problems, see our [tutorial page](#).

By submitting a pull request, you are affirming the Developer's Certificate of Origin at the end of this file.

3.1 Quick Summary

- We encourage you to *join the hIPPYlib organization* and create development branches off `hippylib:master`.
- Please follow the *developer guidelines*, in particular with regards to documentation and code styling.

- Pull requests should be issued toward `hippylib:master`. Make sure to check the items off the *Pull Request Checklist*.
- After approval, hIPPYlib developers merge the PR in `hippylib:master`.
- Don't hesitate to *contact us* if you have any questions.

3.2 GitHub Workflow

The GitHub organization, <https://github.com/hippylib>, is the main developer hub for the hIPPYlib project.

If you plan to make contributions or will like to stay up-to-date with changes in the code, **we strongly encourage you to join the hIPPYlib organization**.

This will simplify the workflow (by providing you additional permissions), and will allow us to reach you directly with project announcements.

3.2.1 hIPPYlib Organization

- Before you can start, you need a GitHub account, here are a few suggestions:
 - Create the account at: github.com/join.
 - For easy identification, please add your name and maybe a picture of you at: <https://github.com/settings/profile>.
 - To receive notification, set a primary email at: <https://github.com/settings/emails>.
 - For password-less pull/push over SSH, add your SSH keys at: <https://github.com/settings/keys>.
- *Contact us* for an invitation to join the hIPPYlib GitHub organization.
- You should receive an invitation email, which you can directly accept. Alternatively, *after logging into GitHub*, you can accept the invitation at the top of <https://github.com/hippylib>.
- Consider making your membership public by going to <https://github.com/orgs/hippylib/people> and clicking on the organization visibility dropdown next to your name.
- Project discussions and announcements will be posted at <https://github.com/orgs/hippylib/teams/everyone>.
- The hIPPYlib source code is in the [hippylib](#) repository.
- The website is in the [web](#) repository.

3.2.2 New Feature Development

- A new feature should be important enough that at least one person, the proposer, is willing to work on it and be its champion.
- The proposer creates a branch for the new feature (with suffix `-dev`), off the `master` branch, or another existing feature branch, for example:

```
# Clone assuming you have setup your ssh keys on GitHub:
git clone git@github.com:hippylib/hippylib.git

# Alternatively, clone using the "https" protocol:
git clone https://github.com/hippylib/hippylib.git
```

(continues on next page)

(continued from previous page)

```
# Create a new feature branch starting from "master":
git checkout master
git pull
git checkout -b feature-dev

# Work on "feature-dev", add local commits
# ...

# (One time only) push the branch to github and setup your local
# branch to track the github branch (for "git pull"):
git push -u origin feature-dev
```

- **We prefer that you create the new feature branch as a fork.** To allow hIPPYlib developers to edit the PR, please [enable upstream edits](#).
- The typical feature branch name is `new-feature-dev`, e.g. `optimal_exp_design-dev`. While not frequent in hIPPYlib, other suffixes are possible, e.g. `-fix`, `-doc`, etc.

3.2.3 Developer Guidelines

- *Keep the code lean and as simple as possible*
 - Well-designed simple code is frequently more general and powerful.
 - Lean code base is easier to understand by new collaborators.
 - New features should be added only if they are necessary or generally useful.
 - Code must be compatible with Python 3.
 - When adding new features add an example in the `application` folder and/or a new notebook in the `tutorial` folder.
 - The preferred way to export solutions for visualization in paraview is using `dl.XDMFFile`
- *Keep the code general and reasonably efficient*
 - Main goal is fast prototyping for research.
 - When in doubt, generality wins over efficiency.
 - Respect the needs of different users (current and/or future).
- *Keep things separate and logically organized*
 - General usage features go in hIPPYlib (implemented in as much generality as possible), non-general features go into external apps/projects.
 - Inside hIPPYlib, compartmentalize between modeling, algorithms, utils, etc.
 - Contributions that are project-specific or have external dependencies are allowed (if they are of broader interest), but should be `#ifdef`-ed and not change the code by default.
- *Code specifics*
 - All significant new classes, methods and functions have sphinx-style documentation in source comments.
 - Code styling should resemble existing code.
 - When manually resolving conflicts during a merge, make sure to mention the conflicted files in the commit message.

3.2.4 Pull Requests

- When your branch is ready for other developers to review / comment on the code, create a pull request towards `hippylib:master`.

- Pull request typically have titles like:

`Description [new-feature-dev]`

for example:

`Bayesian Optimal Design of Experiments [oed-dev]`

Note the branch name suffix (in square brackets).

- Titles may contain a prefix in square brackets to emphasize the type of PR. Common choices are: `[DON'T MERGE]`, `[WIP]` and `[DISCUSS]`, for example:

`[DISCUSS] Bayesian Optimal Design of Experiments [oed-dev]`

- Add a description, appropriate labels and assign yourself to the PR. The hIPPYlib team will add reviewers as appropriate.
- List outstanding TODO items in the description.
- Track the Travis CI *continuous integration* builds at the end of the PR. These should run clean, so address any errors as soon as possible.

3.2.5 Pull Request Checklist

Before a PR can be merged, it should satisfy the following:

- ☐ CI runs without errors.
- ☐ Update `CHANGELOG`:
 - ☐ Is this a new feature users need to be aware of? New or updated application or tutorial?
 - ☐ Does it make sense to create a new section in the `CHANGELOG` to group with other related features?
- ☐ New examples/applications/tutorials:
 - ☐ All new examples/applications/tutorials run as expected.
 - ☐ Add a *fast version* of the example/application/tutorial to Travis CI
- ☐ New capability:
 - ☐ All significant new classes, methods and functions have sphinx-style documentation in source comments.
 - ☐ Add new examples/applications/tutorials to highlight the new capability.
 - ☐ For new classes, functions, or modules, edit the corresponding `.rst` file in the `doc` folder.
 - ☐ If this is a major new feature, consider mentioning in the short summary inside `README` (*rare*).
 - ☐ If this is a C++ extension, the `package_data` dictionary in `setup.py` should include new files.

3.3 Automated Testing

We use Travis CI to drive the default tests on the `master` and `feature` branches. See the `.travis` file and the logs at <https://travis-ci.org/hi/hippylib>.

Testing using Travis CI should be kept lightweight, as there is a 50 minute time constraint on jobs.

- Tests on the `master` branch are triggered whenever a push is issued on this branch.

3.4 Contact Information

- Contact the hIPPYlib team by posting to the [GitHub issue tracker](#). Please perform a search to make sure your question has not been answered already.

3.5 Slack channel

The hIPPYlib slack channel is a good resource to request and receive help with using hIPPYlib. Everyone is invited to read and take part in discussions. Discussions about development of new features in hIPPYlib also take place here. You can join our Slack community by filling in [this form](#).

3.6 Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

- (a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or
- (b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or
- (c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.
- (d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

Acknowledgement: We thank the [MFEM team](#) for allowing us to use their contributing guidelines file as template.

4.1 hippylib package

4.1.1 Subpackages

hippylib.modeling package

Submodules

hippylib.modeling.PDEProblem module

class hippylib.modeling.PDEProblem.**PDEProblem**

Bases: object

Consider the PDE problem: Given m , find u such that

$$F(u, m, p) = (f(u, m), p) = 0, \quad \forall p.$$

Here F is linear in p , but it may be non linear in u and m .

apply_ij (i, j, dir, out)

Given u, m, p ; compute $\delta_{ij}F(u, m, p; \hat{i}, \tilde{j})$ in the direction $\tilde{j} = dir, \forall \hat{i}$.

apply_ijk ($i, j, k, x, jdir, kdir, out$)

Given $x = [u, a, p]$; compute $\delta_{ijk}F(u, a, p; \hat{i}, \tilde{j}, \tilde{k})$ in the direction $(\tilde{j}, \tilde{k}) = (jdir, kdir), \forall \hat{i}$.

evalGradientParameter (x, out)

Given u, m, p ; evaluate $\delta_m F(u, m, p; \hat{m}), \forall \hat{m}$.

generate_parameter ()

Return a vector in the shape of the parameter.

generate_state ()

Return a vector in the shape of the state.

init_parameter (*m*)

Initialize the parameter.

setLinearizationPoint (*x*, *gauss_newton_approx*)

Set the values of the state and parameter for the incremental forward and adjoint solvers. Set whether Gauss Newton approximation of the Hessian should be used.

solveAdj (*adj*, *x*, *adj_rhs*)

Solve the linear adjoint problem: Given *m*, *u*; find *p* such that

$$\delta_u F(u, m, p; \hat{u}) = 0, \quad \forall \hat{u}.$$

solveFwd (*state*, *x*)

Solve the possibly nonlinear forward problem: Given *m*, find *u* such that

$$\delta_p F(u, m, p; \hat{p}) = 0, \quad \forall \hat{p}.$$

solveIncremental (*out*, *rhs*, *is_adj*)

If *is_adj* = False:

Solve the forward incremental system: Given *u*, *m*, find \tilde{u} such that

$$\delta_{pu} F(u, m, p; \hat{p}, \tilde{u}) = \text{rhs}, \quad \forall \hat{p}.$$

If *is_adj* = True:

Solve the adjoint incremental system: Given *u*, *m*, find \tilde{p} such that

$$\delta_{up} F(u, m, p; \hat{u}, \tilde{p}) = \text{rhs}, \quad \forall \hat{u}.$$

class hippylib.modeling.PDEProblem.**PDEVariationalProblem** (*Vh*, *varf_handler*,
bc, *bc0*,
is_fwd_linear=False)

Bases: *hippylib.modeling.PDEProblem.PDEProblem*

_createLUSolver ()

apply_ij (*i*, *j*, *dir*, *out*)

Given *u*, *m*, *p*; compute $\delta_{ij} F(u, m, p; \hat{i}, \tilde{j})$ in the direction $\tilde{j} = \text{dir}$, $\forall \hat{i}$.

apply_ijk (*i*, *j*, *k*, *x*, *jdir*, *kdir*, *out*)

Given $\mathbf{x} = [u, a, p]$; compute $\delta_{ijk} F(u, a, p; \hat{i}, \tilde{j}, \tilde{k})$ in the direction $(\tilde{j}, \tilde{k}) = (jdir, kdir)$, $\forall \hat{i}$.

evalGradientParameter (*x*, *out*)

Given *u*, *m*, *p*; evaluate $\delta_m F(u, m, p; \hat{m})$, $\forall \hat{m}$.

generate_parameter ()

Return a vector in the shape of the parameter.

generate_state ()

Return a vector in the shape of the state.

init_parameter (*m*)

Initialize the parameter.

setLinearizationPoint (*x*, *gauss_newton_approx*)

Set the values of the state and parameter for the incremental forward and adjoint solvers.

solveAdj (*adj*, *x*, *adj_rhs*)

Solve the linear adjoint problem: Given m , u ; find p such that

$$\delta_u F(u, m, p; \hat{u}) = 0, \quad \forall \hat{u}.$$

solveFwd (*state*, *x*)

Solve the possibly nonlinear forward problem: Given m , find u such that

$$\delta_p F(u, m, p; \hat{p}) = 0, \quad \forall \hat{p}.$$

solveIncremental (*out*, *rhs*, *is_adj*)

If *is_adj* == False:

Solve the forward incremental system: Given u , m , find \tilde{u} such that

$$\delta_{pu} F(u, m, p; \hat{p}, \tilde{u}) = \text{rhs}, \quad \forall \hat{p}.$$

If *is_adj* == True:

Solve the adjoint incremental system: Given u , m , find \tilde{p} such that

$$\delta_{up} F(u, m, p; \hat{u}, \tilde{p}) = \text{rhs}, \quad \forall \hat{u}.$$

hippylib.modeling.expression module

hippylib.modeling.misfit module

class hippylib.modeling.misfit.**ContinuousStateObservation** (*Vh*, *dX*, *bcs*, *form=None*)

Bases: *hippylib.modeling.misfit.Misfit*

This class implements continuous state observations in a subdomain $X \subset \Omega$ or $X \subset \partial\Omega$.

Constructor:

Vh: the finite element space for the state variable.

dX: the integrator on subdomain X where observation are presents. E.g. *dX* = *ufl.dx* means observation on all Ω and *dX* = *ufl.ds* means observations on all $\partial\Omega$.

bcs: If the forward problem imposes Dirichlet boundary conditions $u = u_D$ on Γ_D ; *bcs* is a list of *dolfin.DirichletBC* object that prescribes homogeneous Dirichlet conditions $u = 0$ on Γ_D .

form: if *form* = *None* we compute the $L^2(X)$ misfit: $\int_X (u - u_d)^2 dX$, otherwise the integrand specified in the given *form* will be used.

apply_ij (*i*, *j*, *dir*, *out*)

Apply the second variation δ_{ij} (*i*, *j* = STATE, PARAMETER) of the cost in direction *dir*.

cost (*x*)

Given *x* evaluate the cost functional. Only the state *u* and (possibly) the parameter *m* are accessed.

grad (*i, x, out*)

Given the state and the paramter in *x*, compute the partial gradient of the misfit functional in with respect to the state (*i* == STATE) or with respect to the parameter (*i* == PARAMETER).

setLinearizationPoint (*x, gauss_newton_approx=False*)

Set the point for linearization.

Inputs:

x=[*u, m, p*] - linearization point

gauss_newton_approx (bool) - whether to use Gauss Newton approximation

class hippylib.modeling.misfit.**Misfit**

Bases: object

Abstract class to model the misfit component of the cost functional. In the following *x* will denote the variable [*u, m, p*], denoting respectively the state *u*, the parameter *m*, and the adjoint variable *p*.

The methods in the class misfit will usually access the state *u* and possibly the parameter *m*. The adjoint variables will never be accessed.

apply_ij (*i, j, dir, out*)

Apply the second variation δ_{ij} (*i, j* = STATE, PARAMETER) of the cost in direction *dir*.

cost (*x*)

Given *x* evaluate the cost functional. Only the state *u* and (possibly) the parameter *m* are accessed.

grad (*i, x, out*)

Given the state and the paramter in *x*, compute the partial gradient of the misfit functional in with respect to the state (*i* == STATE) or with respect to the parameter (*i* == PARAMETER).

setLinearizationPoint (*x, gauss_newton_approx=False*)

Set the point for linearization.

Inputs:

x=[*u, m, p*] - linearization point

gauss_newton_approx (bool) - whether to use Gauss Newton approximation

class hippylib.modeling.misfit.**MultiPointwiseStateObservation** (*Vh, obs_points, Mpar*)

Bases: *hippylib.modeling.misfit.Misfit*

This class implements pointwise state observations at given locations. A multiplicative Gamma(M,M) noise model is assumed

Constructor:

Vh is the finite element space for the state variable

obs_points is a 2D array number of points by geometric dimensions that stores the location of the observations.

apply_ij (*i, j, dir, out*)

Apply the second variation δ_{ij} (*i, j* = STATE, PARAMETER) of the cost in direction *dir*.

cost (*x*)

Given *x* evaluate the cost functional. Only the state *u* and (possibly) the parameter *m* are accessed.

grad (*i*, *x*, *out*)

Given the state and the paramter in *x*, compute the partial gradient of the misfit functional in with respect to the state (*i* == STATE) or with respect to the parameter (*i* == PARAMETER).

setLinearizationPoint (*x*, *gauss_newton_approx=False*)

Set the point for linearization.

Inputs:

x=[*u*, *m*, *p*] - linearization point

gauss_newton_approx (bool) - whether to use Gauss Newton approximation

class hippylib.modeling.misfit.**MultiStateMisfit** (*misfits*)

Bases: [hippylib.modeling.misfit.Misfit](#)

append (*misfit*)

apply_ij (*i*, *j*, *dir*, *out*)

Apply the second variation δ_{ij} (*i*, *j* = STATE, PARAMETER) of the cost in direction *dir*.

cost (*x*)

Given *x* evaluate the cost functional. Only the state *u* and (possibly) the parameter *m* are accessed.

grad (*i*, *x*, *out*)

Given the state and the paramter in *x*, compute the partial gradient of the misfit functional in with respect to the state (*i* == STATE) or with respect to the parameter (*i* == PARAMETER).

setLinearizationPoint (*x*, *gauss_newton_approx=False*)

Set the point for linearization.

Inputs:

x=[*u*, *m*, *p*] - linearization point

gauss_newton_approx (bool) - whether to use Gauss Newton approximation

class hippylib.modeling.misfit.**PointwiseStateObservation** (*Vh*, *obs_points*)

Bases: [hippylib.modeling.misfit.Misfit](#)

This class implements pointwise state observations at given locations. It assumes that the state variable is a scalar function.

Constructor:

Vh is the finite element space for the state variable

obs_points is a 2D array number of points by geometric dimensions that stores the location of the observations.

apply_ij (*i*, *j*, *dir*, *out*)

Apply the second variation δ_{ij} (*i*, *j* = STATE, PARAMETER) of the cost in direction *dir*.

cost (*x*)

Given *x* evaluate the cost functional. Only the state *u* and (possibly) the parameter *m* are accessed.

grad (*i*, *x*, *out*)

Given the state and the paramter in *x*, compute the partial gradient of the misfit functional in with respect to the state (*i* == STATE) or with respect to the parameter (*i* == PARAMETER).

setLinearizationPoint (*x*, *gauss_newton_approx=False*)

Set the point for linearization.

Inputs:

$x=[u, m, p]$ - linearization point

`gauss_newton_approx` (bool) - whether to use Gauss Newton approximation

hippylib.modeling.model module

class `hippylib.modeling.model.Model` (*problem, prior, misfit*)

Bases: `object`

This class contains the full description of the inverse problem. As inputs it takes a `PDEProblem` object, a `Prior` object, and a `Misfit` object.

In the following we will denote with

- u the state variable
- m the (model) parameter variable
- p the adjoint variable

Create a model given:

- `problem`: the description of the forward/adjoint problem and all the sensitivities
- `prior`: the prior component of the cost functional
- `misfit`: the misfit component of the cost functional

Rsolver ()

Return an object `Rsolver` that is a suitable solver for the regularization operator R .

The solver object should implement the method `Rsolver.solve(z, r)` such that $Rz \approx r$.

applyC (*dm, out*)

Apply the C block of the Hessian to a (incremental) parameter variable, i.e. $out = Cdm$

Parameters:

- `dm` the (incremental) parameter variable
- `out` the action of the C block on `dm`

Note: This routine assumes that `out` has the correct shape.

applyCt (*dp, out*)

Apply the transpose of the C block of the Hessian to a (incremental) adjoint variable. $out = C^t dp$

Parameters:

- `dp` the (incremental) adjoint variable
- `out` the action of the C^T block on `dp`

..note:: This routine assumes that `out` has the correct shape.

applyR (*dm, out*)

Apply the regularization R to a (incremental) parameter variable. $out = Rdm$

Parameters:

- `dm` the (incremental) parameter variable
- `out` the action of R on `dm`

Note: This routine assumes that `out` has the correct shape.

applyWmm (*dm, out*)

Apply the W_{mm} block of the Hessian to a (incremental) parameter variable. $out = W_{mm}dm$

Parameters:

- `dm` the (incremental) parameter variable
- `out` the action of the W_{mm} on block `dm`

Note: This routine assumes that `out` has the correct shape.

applyWmu (*du, out*)

Apply the W_{mu} block of the Hessian to a (incremental) state variable. $out = W_{mu}du$

Parameters:

- `du` the (incremental) state variable
- `out` the action of the W_{mu} block on `du`

Note: This routine assumes that `out` has the correct shape.

applyWum (*dm, out*)

Apply the W_{um} block of the Hessian to a (incremental) parameter variable. $out = W_{um}dm$

Parameters:

- `dm` the (incremental) parameter variable
- `out` the action of the W_{um} block on `du`

Note: This routine assumes that `out` has the correct shape.

applyWuu (*du, out*)

Apply the W_{uu} block of the Hessian to a (incremental) state variable. $out = W_{uu}du$

Parameters:

- `du` the (incremental) state variable
- `out` the action of the W_{uu} block on `du`

Note: This routine assumes that `out` has the correct shape.

apply_ij (*i, j, d, out*)

cost (*x*)

Given the list `x = [u, m, p]` which describes the state, parameter, and adjoint variable compute the cost functional as the sum of the misfit functional and the regularization functional.

Return the list [cost functional, regularization functional, misfit functional]

Note: `p` is not needed to compute the cost functional

evalGradientParameter (*x, mg, misfit_only=False*)

Evaluate the gradient for the variational parameter equation at the point $x = [u, m, p]$.

Parameters:

- $x = [u, m, p]$ the point at which to evaluate the gradient.
- `mg` the variational gradient ($g, mtest$), `mtest` being a test function in the parameter space (Output parameter)

Returns the norm of the gradient in the correct inner product $g_{norm} = \text{sqrt}(g, g)$

generate_vector (*component='ALL'*)

By default, return the list $[u, m, p]$ where:

- `u` is any object that describes the state variable
- `m` is a `dolfin.Vector` object that describes the parameter variable. (Needs to support linear algebra operations)
- `p` is any object that describes the adjoint variable

If `component = STATE` return only `u`

If `component = PARAMETER` return only `m`

If `component = ADJOINT` return only `p`

init_parameter (*m*)

Reshape `m` so that it is compatible with the parameter variable

setPointForHessianEvaluations (*x, gauss_newton_approx=False*)

Specify the point $x = [u, m, p]$ at which the Hessian operator (or the Gauss-Newton approximation) needs to be evaluated.

Parameters:

- $x = [u, m, p]$: the point at which the Hessian or its Gauss-Newton approximation needs to be evaluated.
- `gauss_newton_approx` (bool): whether to use Gauss-Newton approximation (default: use Newton)

Note: This routine should either:

- simply store a copy of `x` and evaluate action of blocks of the Hessian on the fly
 - or partially precompute the block of the hessian (if feasible)
-

solveAdj (*out, x*)

Solve the linear adjoint problem.

Parameters:

- `out`: is the solution of the adjoint problem (i.e. the adjoint `p`) (Output parameter)
- $x = [u, m, p]$ provides
 - 1) the parameter variable `m` for assembling the adjoint operator

- 2) the state variable u for assembling the adjoint right hand side

Note: p is not accessed

solveAdjIncremental (*sol, rhs*)

Solve the incremental adjoint problem for a given right-hand side

Parameters:

- *sol* the solution of the incremental adjoint problem (Output)
- *rhs* the right hand side of the linear system

solveFwd (*out, x*)

Solve the (possibly non-linear) forward problem.

Parameters:

- *out*: is the solution of the forward problem (i.e. the state) (Output parameters)
- $x = [u, m, p]$ provides
 - 1) the parameter variable m for the solution of the forward problem
 - 2) the initial guess u if the forward problem is non-linear

Note: p is not accessed.

solveFwdIncremental (*sol, rhs*)

Solve the linearized (incremental) forward problem for a given right-hand side

Parameters:

- *sol* the solution of the linearized forward problem (Output)
- *rhs* the right hand side of the linear system

hippylib.modeling.modelVerify module

`hippylib.modeling.modelVerify.modelVerify(model, m0, is_quadratic=False, mis_fit_only=False, verbose=True, eps=None)`

Verify the reduced Gradient and the Hessian of a model. It will produce two loglog plots of the finite difference checks for the gradient and for the Hessian. It will also check for symmetry of the Hessian.

`hippylib.modeling.modelVerify.modelVerifyPlotErrors(is_quadratic, eps, err_grad, err_H)`

hippylib.modeling.pointwiseObservation module

`hippylib.modeling.pointwiseObservation.assemblePointwiseObservation(Vh, targets, prune_and_sort=False)`

Assemble the pointwise observation matrix:

Inputs

- *Vh*: FEniCS finite element space.

- targets: observation points (numpy array).

```
hippylib.modeling.pointwiseObservation.exportPointwiseObservation(Vh, B, data,
                                                                    fname, var-
                                                                    name='observation')
```

This function writes a VTK PolyData file to visualize pointwise data.

Inputs:

- B: observation operator.
- mesh: mesh.
- data: dolfin.Vector containing the data.
- fname: filename for the file to export (without extension).
- varname: name of the variable for the .vtk file.

```
hippylib.modeling.pointwiseObservation.write_vtk(points, data, fname, var-
                                                  name='observation')
```

This function writes a VTK PolyData file to visualize pointwise data.

Inputs:

- points: locations of the points (numpy array of size equal to number of points times space dimension).
- data: pointwise values (numpy array of size equal to number of points).
- fname: filename for the .vtk file to export.
- varname: name of the variable for the .vtk file.

hippylib.modeling.posterior module

```
class hippylib.modeling.posterior.GaussianLRPosterior(prior, d, U, mean=None)
```

Bases: object

Class for the low rank Gaussian Approximation of the Posterior. This class provides functionality for approximate Hessian apply, solve, and Gaussian sampling based on the low rank factorization of the Hessian.

In particular if d and U are the dominant eigenpairs of H_{misfit} $U[:, i] = d[i]RU[:, i]$ then we have:

- low rank Hessian apply: $y = (R + RUDU^T)x$.
- low rank Hessian solve: $y = (R^{-1} - U(I + D^{-1})^{-1}U^T)x$.
- low rank Hessian Gaussian sampling: $y = (I - USU^T)x$, where $S = I - (I + D)^{-1/2}$ and $x \sim \mathcal{N}(0, R^{-1})$.

Construct the Gaussian approximation of the posterior. Input: - prior: the prior mode. - d: the dominant generalized eigenvalues of the Hessian misfit. - U: the dominant generalized eigenvector of the Hessian misfit $U^T RU = I$. - mean: the MAP point.

```
_sample_given_prior(s_prior, s_post)
```

```
_sample_given_white_noise(noise, s_prior, s_post)
```

```
cost(m)
```

```
init_vector(x, dim)
```

Initialize a vector x to be compatible with the range/domain of H . If `dim == "noise"` initialize x to be compatible with the size of white noise used for sampling.

```
klDistanceFromPrior(sub_comp=False)
```


pointwise_variance (***kwargs*)

Compute/estimate the pointwise variance of the posterior, prior distribution and the pointwise variance reduction informed by the data.

See `_Prior.pointwise_variance` for more details.

sample (**args, **kwargs*)

possible calls:

1) `sample(s_prior, s_post, add_mean=True)`

Given a prior sample `s_prior` compute a sample `s_post` from the posterior.

- `s_prior` is a sample from the prior centered at 0 (input).
- `s_post` is a sample from the posterior (output).
- if `add_mean=True` (default) then the samples will be centered at the map point.

2) `sample(noise, s_prior, s_post, add_mean=True)`

Given $\text{noise} \sim \mathcal{N}(0, I)$ compute a sample `s_prior` from the prior and `s_post` from the posterior.

- `noise` is a realization of white noise (input).
- `s_prior` is a sample from the prior (output).
- `s_post` is a sample from the posterior.
- if `add_mean=True` (default) then the prior and posterior samples will be centered at the respective means.

trace (***kwargs*)

Compute/estimate the trace of the posterior, prior distribution and the trace of the data informed correction.

See `_Prior.trace` for more details.

trace_update ()

class `hippylib.modeling.posterior.LowRankHessian` (*prior, d, U*)

Bases: `object`

Operator that represents the action of the low rank approximation of the Hessian and of its inverse.

init_vector (*x, dim*)

inner (*x, y*)

mult (*x, y*)

solve (*sol, rhs*)

class `hippylib.modeling.posterior.LowRankPosteriorSampler` (*prior, d, U*)

Bases: `object`

Object to sample from the low rank approximation of the posterior.

$$y = (I - USU^T)x,$$

where

$$S = I - (I + D)^{-1/2}, x \sim \mathcal{N}(0, R^{-1}).$$

init_vector (*x, dim*)

sample (*noise, s*)

hippylib.modeling.prior module

`hippylib.modeling.prior.BiLaplacianComputeCoefficients` (*sigma2, rho, ndim*)

This class is responsible to compute the parameters gamma and delta for the BiLaplacianPrior given the marginal variance sigma2 and correlation length rho. ndim is the dimension of the domain 2D or 3D

`hippylib.modeling.prior.BiLaplacianPrior` (*Vh, gamma, delta, Theta=None, mean=None, rel_tol=1e-12, max_iter=1000, robin_bc=False*)

This function construct an instance of `SqrtPrecisionPDE_Prior` with covariance matrix $C = (\delta I + \gamma \operatorname{div} \Theta \nabla)^{-2}$.

The magnitude of $\delta\gamma$ governs the variance of the samples, while the ratio $\frac{\gamma}{\delta}$ governs the correlation length.

Here Θ is a SPD tensor that models anisotropy in the covariance kernel.

Input:

- *Vh*: the finite element space for the parameter
- *gamma* and *delta*: the coefficient in the PDE (floats, `dl.Constant`, `dl.Expression`, or `dl.Function`)
- *Theta*: the SPD tensor for anisotropic diffusion of the PDE
- *mean*: the prior mean
- *rel_tol*: relative tolerance for solving linear systems involving covariance matrix
- *max_iter*: maximum number of iterations for solving linear systems involving covariance matrix
- *robin_bc*: whether to use Robin boundary condition to remove boundary artifacts

class `hippylib.modeling.prior.GaussianRealPrior` (*Vh, covariance, mean=None*)

Bases: `hippylib.modeling.prior._Prior`

This class implements a finite-dimensional Gaussian prior, $\mathcal{N}(\mathbf{m}, \mathbf{C})$, where \mathbf{m} is the mean of the Gaussian distribution, and \mathbf{C} is its covariance. The underlying finite element space is assumed to be the “R” space.

Constructor

Inputs: - *Vh*: Finite element space on which the prior is

defined. Must be the Real space with one global degree of freedom

- **covariance:** The covariance of the prior. Must be a `numpy.ndarray` of appropriate size
- **:code:‘mean’(optional):** Mean of the prior distribution. Must be of type `dolfin.Vector()`

init_vector (*x, dim*)

Initialize a vector *x* to be compatible with the range/domain of *R*.

If *dim* == “noise” initialize *x* to be compatible with the size of white noise used for sampling.

sample (*noise, s, add_mean=True*)

Given $\text{noise} \sim \mathcal{N}(0, I)$ compute a sample *s* from the prior.

If *add_mean* == True add the prior mean value to *s*.

class `hippylib.modeling.prior.LaplacianPrior` (*Vh, gamma, delta, mean=None, rel_tol=1e-12, max_iter=100*)

Bases: `hippylib.modeling.prior._Prior`

This class implements a prior model with covariance matrix $C = (\delta I - \gamma \Delta)^{-1}$.

The magnitude of γ governs the variance of the samples, while the ratio $\frac{\gamma}{\delta}$ governs the correlation length.

Note: C is a trace class operator only in 1D while it is not a valid prior in 2D and 3D.

Construct the prior model. Input:

- `Vh`: the finite element space for the parameter
- `gamma` and `delta`: the coefficient in the PDE
- `Theta`: the SPD tensor for anisotropic diffusion of the PDE
- `mean`: the prior mean

init_vector (`x`, `dim`)

Initialize a vector `x` to be compatible with the range/domain of R .

If `dim == "noise"` initialize `x` to be compatible with the size of white noise used for sampling.

sample (`noise`, `s`, `add_mean=True`)

Given $\text{noise} \sim \mathcal{N}(0, I)$ compute a sample `s` from the prior.

If `add_mean == True` add the prior mean value to `s`.

```
hippylib.modeling.prior.MollifiedBiLaplacianPrior(Vh, gamma, delta, locations, m_true, Theta=None,
                                                    pen=10.0, order=2, rel_tol=1e-12,
                                                    max_iter=1000)
```

This function construct an instance of :code".SqrtPrecisionPDE_Prior" with covariance matrix $C = ([\delta + \text{pen} \sum_i m(x - x_i)]I + \gamma \text{div } \Theta \nabla)^{-2}$,

where

- Θ is a SPD tensor that models anisotropy in the covariance kernel.
- $x_i (i = 1, \dots, n)$ are points where we assume to know exactly the value of the parameter (i.e., $m(x_i) = m_{\text{true}}(x_i)$ for $i = 1, \dots, n$).
- m is the mollifier function: $m(x - x_i) = \exp\left(-\left[\frac{\gamma}{\delta}\|x - x_i\|_{\Theta^{-1}}\right]^{\text{order}}\right)$.
- `pen` is a penalization parameter.

The magnitude of $\delta\gamma$ governs the variance of the samples, while the ratio $\frac{\gamma}{\delta}$ governs the correlation length.

The prior mean is computed by solving

$$\left([\delta + \sum_i m(x - x_i)]I + \gamma \text{div } \Theta \nabla\right) m = \sum_i m(x - x_i) m_{\text{true}}.$$

Input:

- `Vh`: the finite element space for the parameter
- `gamma` and `delta`: the coefficients in the PDE
- `locations`: the points x_i at which we assume to know the true value of the parameter
- `m_true`: the true model
- `Theta`: the SPD tensor for anisotropic diffusion of the PDE
- `pen`: a penalization parameter for the mollifier

```
class hippylib.modeling.prior.SqrtPrecisionPDE_Prior(Vh,
                                                    sqrt_precision_varf_handler,
                                                    mean=None, rel_tol=1e-12,
                                                    max_iter=1000)
```

Bases: `hippylib.modeling.prior._Prior`

This class implement a prior model with covariance matrix $C = A^{-1}MA^{-1}$, where A is the finite element matrix arising from discretization of `sqrt_precision_varf_handler`

Construct the prior model. Input:

- Vh: the finite element space for the parameter
- **code:sqrt_precision_varf_handler** the PDE representation of the sqrt of the covariance operator
- mean: the prior mean

```
init_vector(x, dim)
```

Initialize a vector x to be compatible with the range/domain of R.

If dim == "noise" initialize x to be compatible with the size of white noise used for sampling.

```
sample(noise, s, add_mean=True)
```

Given noise $\sim \mathcal{N}(0, I)$ compute a sample s from the prior.

If add_mean == True add the prior mean value to s.

```
class hippylib.modeling.prior._BilaplacianR(A, Msolver)
```

Bases: object

Operator that represent the action of the regularization/precision matrix for the Bilaplacian prior.

```
init_vector(x, dim)
```

```
mpi_comm()
```

```
mult(x, y)
```

```
class hippylib.modeling.prior._BilaplacianRsolver(Asolver, M)
```

Bases: object

Operator that represent the action of the inverse the regularization/precision matrix for the Bilaplacian prior.

```
init_vector(x, dim)
```

```
solve(x, b)
```

```
class hippylib.modeling.prior._Prior
```

Bases: object

Abstract class to describe the prior model. Concrete instances of a `_Prior` class should expose the following attributes and methods.

Attributes:

- R: an operator to apply the regularization/precision operator.
- Rsolver: an operator to apply the inverse of the regularization/precision operator.
- M: the mass matrix in the control space.
- mean: the prior mean.

Methods:

- `init_vector(self, x, dim)`: Initialize a vector x to be compatible with the range/domain of R If dim == "noise" initialize x to be compatible with the size of white noise used for sampling.

- `sample(self, noise, s, add_mean=True)`: Given $\text{noise} \sim \mathcal{N}(0, I)$ compute a sample s from the prior. If `add_mean==True` add the prior mean value to s .

`cost(m)`

`getHessianPreconditioner()`

Return the preconditioner for Newton-CG

`grad(m, out)`

`init_vector(x, dim)`

`pointwise_variance(method, k=1000000, r=200)`

Compute/estimate the prior pointwise variance.

- If `method=="Exact"` we compute the diagonal entries of R^{-1} entry by entry. This requires to solve n linear system in R (not scalable, but ok for illustration purposes).

`sample(noise, s, add_mean=True)`

`trace(method='Exact', tol=0.1, min_iter=20, max_iter=100, r=200)`

Compute/estimate the trace of the prior covariance operator.

- If `method=="Exact"` we compute the trace exactly by summing the diagonal entries of $R^{-1}M$. This requires to solve n linear system in R (not scalable, but ok for illustration purposes).
- If `method=="Estimator"` use the trace estimator algorithms implemented in the class `TraceEstimator`. `tol` is a relative bound on the estimator standard deviation. In particular, we used enough samples in the Estimator such that the standard deviation of the estimator is less than `tol*tr(Prior)`. `min_iter` and `max_iter` are the lower and upper bound on the number of samples to be used for the estimation of the trace.

class `hippylib.modeling.prior._RinvM(Rsolver, M)`

Bases: `object`

Operator that models the action of $R^{-1}M$. It is used in the randomized trace estimator.

`init_vector(x, dim)`

`mult(x, y)`

hippylib.modeling.reducedHessian module

class `hippylib.modeling.reducedHessian.FDHessian(model, m0, h, misfit_only=False)`

Bases: `object`

This class implements matrix free application of the reduced Hessian operator. The constructor takes the following parameters:

- `model`: the object which contains the description of the problem.
- `m0`: the value of the parameter at which the Hessian needs to be evaluated.
- `h`: the mesh size for FD.
- `misfit_only`: a boolean flag that describes whenever the full Hessian or only the misfit component of the Hessian is used.

Type `help(Template)` for more information on which methods model should implement.

Construct the reduced Hessian Operator

init_vector (*x*, *dim*)

Reshape the Vector *x* so that it is compatible with the reduced Hessian operator.

Parameters:

- *x*: the vector to reshape
- *dim*: if 0 then *x* will be reshaped to be compatible with the range of the reduced Hessian, if 1 then *x* will be reshaped to be compatible with the domain of the reduced Hessian

Note: Since the reduced Hessian is a self adjoint operator, the range and the domain is the same. Either way, we choosed to add the parameter *dim* for consistency with the interface of `Matrix` in `dolfin`.

inner (*x*, *y*)

Perform the inner product between *x* and *y* in the norm induced by the reduced Hessian H , $(x, y)_H = x'Hy$.

mult (*x*, *y*)

Apply the reduced Hessian (or the Gauss-Newton approximation) to the vector *x*. Return the result in *y*.

class `hippylib.modeling.reducedHessian.ReducedHessian` (*model*, *misfit_only=False*)

Bases: `object`

This class implements matrix free application of the reduced Hessian operator. The constructor takes the following parameters:

- *model*: the object which contains the description of the problem.
- *misfit_only*: a boolean flag that describes whenever the full Hessian or only the misfit component of the Hessian is used.

Type `help(modelTemplate)` for more information on which methods *model* should implement.

Construct the reduced Hessian Operator

GNHessian (*x*, *y*)

Apply the Gauss-Newton approximation of the reduced Hessian to the vector *x*. Return the result in *y*.

TrueHessian (*x*, *y*)

Apply the the reduced Hessian to the vector *x*. Return the result in *y*.

init_vector (*x*, *dim*)

Reshape the Vector *x* so that it is compatible with the reduced Hessian operator.

Parameters:

- *x*: the vector to reshape.
- *dim*: if 0 then *x* will be reshaped to be compatible with the range of the reduced Hessian, if 1 then *x* will be reshaped to be compatible with the domain of the reduced Hessian.

Note: Since the reduced Hessian is a self adjoint operator, the range and the domain is the same. Either way, we choosed to add the parameter *dim* for consistency with the interface of `Matrix` in `dolfin`.

inner (*x*, *y*)

Perform the inner product between *x* and *y* in the norm induced by the reduced Hessian H , $(x, y)_H = x'Hy$.

mult (*x*, *y*)

Apply the reduced Hessian (or the Gauss-Newton approximation) to the vector *x*. Return the result in *y*.

hippylib.modeling.timeDependentVector module

```
class hippylib.modeling.timeDependentVector.TimeDependentVector (times,  

                                                                tol=1e-10,  

                                                                mpi_comm=<sphinx.ext.autodoc.import  

                                                                object>)
```

Bases: object

A class to store time dependent vectors. Snapshots are stored/retrieved by specifying the time of the snapshot. Times at which the snapshot are taken must be specified in the constructor.

Constructor:

- `times`: time frame at which snapshots are stored.
- `tol`: tolerance to identify the frame of the snapshot.

axpy (*a, other*)
 Compute $x = x + a \cdot \text{other}$ snapshot per snapshot.

copy ()
 Return a copy of all the time frames and snapshots

initialize (*M, dim*)
 Initialize all the snapshot to be compatible with the range/domain of an operator *M*.

inner (*other*)
 Compute the inner products: $a_i = (\text{self}[i], \text{other}[i])$ for each snapshot.

norm (*time_norm, space_norm*)
 Compute the space-time norm of the snapshot.

retrieve (*u, t*)
 Retrieve snapshot *u* relative to time *t*. If *t* does not belong to the list of time frame an error is raised.

store (*u, t*)
 Store snapshot *u* relative to time *t*. If *t* does not belong to the list of time frame an error is raised.

zero ()
 Zero out each snapshot.

hippylib.modeling.variables module

Enumerator for the variables of the inverse problem: - the STATE, PARAMETER, and ADJOINT variables.

Module contents

hippylib.algorithms package

Submodules

hippylib.algorithms.NewtonCG module

```
hippylib.algorithms.NewtonCG.LS_ParameterList ()  

  Generate a ParameterList for line search globalization. type: LS_ParameterList().showMe() for de-  

  fault values and their descriptions
```

```
class hippylib.algorithms.NewtonCG.ReducedSpaceNewtonCG (model,                param-  
                ters=<hippylib.utils.parameterList.ParameterList  
                object>, callback=None)
```

Bases: object

Inexact Newton-CG method to solve constrained optimization problems in the reduced parameter space. The Newton system is solved inexactly by early termination of CG iterations via Eisenstat-Walker (to prevent over-solving) and Steihaug (to avoid negative curvature) criteria. Globalization is performed using one of the following methods:

- line search (LS) based on the armijo sufficient reduction condition; or
- trust region (TR) based on the prior preconditioned norm of the update direction.

The stopping criterion is based on a control on the norm of the gradient and a control of the inner product between the gradient and the Newton direction.

The user must provide a model that describes the forward problem, cost functionals, and all the derivatives for the gradient and the Hessian.

More specifically the model object should implement following methods:

- `generate_vector()` -> generate the object containing state, parameter, adjoint
- `cost(x)` -> evaluate the cost functional, report regularization part and misfit separately
- `solveFwd(out, x)` -> solve the possibly non linear forward problem
- `solveAdj(out, x)` -> solve the linear adjoint problem
- `evalGradientParameter(x, out)` -> evaluate the gradient of the parameter and compute its norm
- `setPointForHessianEvaluations(x)` -> set the state to perform hessian evaluations
- `solveFwdIncremental(out, rhs)` -> solve the linearized forward problem for a given rhs
- `solveAdjIncremental(out, rhs)` -> solve the linear adjoint problem for a given rhs
- `applyC(dm, out)` -> Compute $out = C_x dm$
- `applyCt(dp, out)` -> Compute $out = C_x^T dp$
- `applyWuu(du, out)` -> Compute $out = (W_{uu})_x du$
- `applyWmu(dm, out)` -> Compute $out = (W_{um})_x dm$
- `applyWmu(du, out)` -> Compute $out = W_{mu} du$
- `applyR(dm, out)` -> Compute $out = R dm$
- `applyWmm(dm, out)` -> Compute $out = W_{mm} dm$
- `Rsolver()` -> A solver for the regularization term

Type `help(Model)` for additional information

Initialize the `ReducedSpaceNewtonCG`. Type `ReducedSpaceNewtonCG_ParameterList().showMe()` for list of default parameters and their descriptions.

Parameters: `model` The model object that describes the inverse problem parameters: (type `ParameterList`, optional) set parameters for inexact Newton CG `callback`: (type function handler with signature `callback(it: int, x: list of dl.Vector): --> None`

optional callback function to be called at the end of each iteration. Takes as input the iteration number, and the list of vectors for the state, parameter, adjoint.

`_solve_ls(x)`

Solve the constrained optimization problem with initial guess `x`.

`_solve_tr(x)`

`solve(x)`

Input: `x = [u, m, p]` represents the initial guess (u and p may be None). `x` will be overwritten on return.

`termination_reasons = ['Maximum number of Iteration reached', 'Norm of the gradient le`

`hippylib.algorithms.NewtonCG.ReducedSpaceNewtonCG_ParameterList()`

Generate a `ParameterList` for `ReducedSpaceNewtonCG`. type: `ReducedSpaceNewtonCG_ParameterList()`.
`showMe()` for default values and their descriptions

`hippylib.algorithms.NewtonCG.TR_ParameterList()`

Generate a `ParameterList` for Trust Region globalization. type: `RT_ParameterList().showMe()` for default values and their descriptions

hippylib.algorithms.bfgs module

class `hippylib.algorithms.bfgs.BFGS(model, parameters=<hippylib.utils.parameterList.ParameterList object>)`

Bases: `object`

Implement BFGS technique with backtracking inexact line search and damped updating See Nocedal & Wright (06), ch.6.2, ch.7.3, ch.18.3

The user must provide a model that describes the forward problem, cost functionals, and all the derivatives for the gradient and the Hessian.

More specifically the model object should implement following methods:

- `generate_vector()` -> generate the object containing state, parameter, adjoint
- `cost(x)` -> evaluate the cost functional, report regularization part and misfit separately
- `solveFwd(out, x)` -> solve the possibly non-linear forward problem
- `solveAdj(out, x)` -> solve the linear adjoint problem
- `evalGradientParameter(x, out)` -> evaluate the gradient of the parameter and compute its norm
- `applyR(dm, out)` -> Compute $out = Rdm$
- `Rsolver()` -> A solver for the regularization term

Type `help(Model)` for additional information

Initialize the BFGS solver. Type `BFGS_ParameterList().showMe()` for default parameters and their description

solve (`x`, `H0inv`, `bounds_xPARAM=None`)

Solve the constrained optimization problem with initial guess `x = [u, m0, p]`.

Note: `u` and `p` may be None.

`x` will be overwritten.

`H0inv`: the initial approximated inverse of the Hessian for the BFGS operator. It has an optional method `update(x)` that will update the operator based on `x = [u, m, p]`.

`bounds_xPARAM`: Bound constraint (list with two entries: min and max). Can be either a scalar value or a `dolfin.Vector`.

Return the solution $[u, m, p]$

termination_reasons = ['Maximum number of Iteration reached', 'Norm of the gradient le

hippylib.algorithms.bfgs.BFGS_ParameterList()

class hippylib.algorithms.bfgs.BFGS_operator(parameters=<hippylib.utils.parameterList.ParameterList object>)

Bases: object

set_H0inv(H0inv)

Set user-defined operator corresponding to H0inv

Input:

H0inv: Fenics operator with method solve()

solve(x, b)

Solve system: $H_{bfgs}x = b$ where H_{bfgs} is the approximation to the Hessian build by BFGS. That is, we apply

$$x = (H_{bfgs})^{-1}b = H_k b$$

where H_k matrix is BFGS approximation to the inverse of the Hessian. Computation done via double-loop algorithm.

Inputs:

x = dolfin.Vector - [out]

b = dolfin.Vector - [in]

update(s, y)

Update BFGS operator with most recent gradient update.

To handle potential break from secant condition, update done via damping

Inputs:

s = dolfin.Vector [in] - corresponds to update in medium parameters.

y = dolfin.Vector [in] - corresponds to update in gradient.

hippylib.algorithms.bfgs.BFGSOperator_ParameterList()

class hippylib.algorithms.bfgs.RescaledIdentity(init_vector=None)

Bases: object

Default operator for H0inv, corresponds to applying $d0I$

init_vector(x, dim)

solve(x, b)

hippylib.algorithms.cgsampler module

class hippylib.algorithms.cgsampler.CGSampler

Bases: object

This class implements the CG sampler algorithm to generate samples from $\mathcal{N}(0, A^{-1})$.

Reference: *Albert Parker and Colin Fox Sampling Gaussian Distributions in Krylov Spaces with Conjugate Gradient SIAM J SCI COMPUT, Vol 34, No. 3 pp. B312-B334*

Construct the solver with default parameters `tolerance = 1e-4`

```
print_level = 0
```

```
verbose = 0
```

sample (*noise*, *s*)

Generate a sample $s \sim \mathcal{N}(0, A^{-1})$.

noise is a `numpy.array` of i.i.d. normal variables used as input. For a fixed realization of noise the algorithm is fully deterministic. The size of noise determine the maximum number of CG iterations.

set_operator (*A*)

Set the operator *A*, such that $x \sim \mathcal{N}(0, A^{-1})$.

Note: *A* is any object that provides the methods `init_vector()` and `mult()`

hippylib.algorithms.cgsolverSteihaug module

```
class hippylib.algorithms.cgsolverSteihaug.CGSolverSteihaug (parameters=<hippylib.utils.parameterList.ParameterList
object>,
comm=<sphinx.ext.autodoc.importer._MockObject
object>)
```

Bases: `object`

Solve the linear system $Ax = b$ using preconditioned conjugate gradient (*B* preconditioner) and the Steihaug stopping criterion:

- reason of termination 0: we reached the maximum number of iterations (no convergence)
- reason of termination 1: we reduced the residual up to the given tolerance (convergence)
- reason of termination 2: we reached a negative direction (premature termination due to not spd matrix)
- reason of termination 3: we reached the boundary of the trust region

The stopping criterion is based on either

- the absolute preconditioned residual norm check: $\|r^*\|_{B^{-1}} < atol$
- the relative preconditioned residual norm check: $\|r^*\|_{B^{-1}} / \|r^0\|_{B^{-1}} < rtol$,

where $r^* = b - Ax^*$ is the residual at convergence and $r^0 = b - Ax^0$ is the initial residual.

The operator *A* is set using the method `set_operator(A)`. *A* must provide the following two methods:

- `A.mult(x, y): y = Ax`
- `A.init_vector(x, dim):` initialize the vector *x* so that it is compatible with the range (*dim* = 0) or the domain (*dim* = 1) of *A*.

The preconditioner *B* is set using the method `set_preconditioner(B)`. *B* must provide the following method: - `B.solve(z, r):` *z* is the action of the preconditioner *B* on the vector *r*

To solve the linear system $Ax = b$ call `self.solve(x, b)`. Here *x* and *b* are assumed to be `dofin.Vector` objects.

Type `CGSolverSteihaug_ParameterList().showMe()` for default parameters and their descriptions

```
reason = ['Maximum Number of Iterations Reached', 'Relative/Absolute residual less than
```

```
set_TR (radius, B_op)
```

```

set_operator ( $A$ )
    Set the operator  $A$ .

set_preconditioner ( $B_{\text{solver}}$ )
    Set the preconditioner  $B$ .

solve ( $x, b$ )
    Solve the linear system  $Ax = b$ 

update_x_with_TR ( $x, \alpha, d$ )

update_x_without_TR ( $x, \alpha, d$ )

```

```

hippylib.algorithms.cgsolverSteihaug.CGSolverSteihaug_ParameterList ()
    Generate a ParameterList for CGSolverSteihaug. Type CGSolverSteihaug_ParameterList ().
    showMe() for default values and their descriptions

```

hippylib.algorithms.linalg module

```

class hippylib.algorithms.linalg.DiagonalOperator ( $d$ )
    Bases: object

    init_vector ( $x, \text{dim}$ )

    inner ( $x, y$ )

    mult ( $x, y$ )

hippylib.algorithms.linalg.GetFromOwnedGid ( $v, \text{gid}$ )

hippylib.algorithms.linalg.MatAtB ( $A, B$ )
    Compute the matrix-matrix product  $A^T B$ .

hippylib.algorithms.linalg.MatMatMult ( $A, B$ )
    Compute the matrix-matrix product  $AB$ .

hippylib.algorithms.linalg.MatPtAP ( $A, P$ )
    Compute the triple matrix product  $P^T A P$ .

class hippylib.algorithms.linalg.Operator2Solver ( $op, \text{mpi\_comm}=<\text{sphinx.ext.autodoc.importer._MockObject}$ 
    object>)
    Bases: object

    init_vector ( $x, \text{dim}$ )

    inner ( $x, y$ )

    solve ( $y, x$ )

hippylib.algorithms.linalg.SetToOwnedGid ( $v, \text{gid}, \text{val}$ )

class hippylib.algorithms.linalg.Solver2Operator ( $S, \text{mpi\_comm}=<\text{sphinx.ext.autodoc.importer._MockObject}$ 
    object>, init_vector=None)
    Bases: object

    init_vector ( $x, \text{dim}$ )

    inner ( $x, y$ )

    mult ( $x, y$ )

hippylib.algorithms.linalg.Transpose ( $A$ )
    Compute the matrix transpose

```

`hippylib.algorithms.linalg.amg_method(amg_type='ml_amg')`

Determine which AMG preconditioner to use. If available use the preconditioner suggested by the user (ML is default). If not available use `petsc_amg`.

`hippylib.algorithms.linalg.estimate_diagonal_inv2(Asolver, k, d)`

An unbiased stochastic estimator for the diagonal of A^{-1} . $d = [\sum_{j=1}^k v_j \cdot A^{-1} v_j] / [\sum_{j=1}^k v_j \cdot v_j]$ where

- v_j are i.i.d. $\mathcal{N}(0, I)$
- \cdot and $/$ represent the element-wise multiplication and division of vectors, respectively.

Reference: *Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad, An estimator for the diagonal of a matrix, Applied Numerical Mathematics, 57 (2007), pp. 1214-1229.*

`hippylib.algorithms.linalg.get_diagonal(A, d)`

Compute the diagonal of the square operator A . Use `Solver2Operator` if A^{-1} is needed.

`hippylib.algorithms.linalg.to_dense(A, mpi_comm=<sphinx.ext.autodoc.importer._MockObject object>)`

Convert a sparse matrix A to dense. For debugging only.

`hippylib.algorithms.linalg.trace(A, mpi_comm=<sphinx.ext.autodoc.importer._MockObject object>)`

Compute the trace of a sparse matrix A .

hippylib.algorithms.lowRankOperator module

class `hippylib.algorithms.lowRankOperator.LowRankOperator(d, U, my_init_vector=None)`

Bases: `object`

This class model the action of a low rank operator $A = UDU^T$. Here D is a diagonal matrix, and the columns of U are orthonormal in some weighted inner-product.

Note: This class only works in serial!

Construct the low rank operator given d and U .

get_diagonal (*diag*)

Compute the diagonal of A .

init_vector (*x*, *dim*)

Initialize x to be compatible with the range ($\text{dim}=0$) or domain ($\text{dim}=1$) of A .

inner (*x*, *y*)

mult (*x*, *y*)

Compute $y = Ax = UDU^T x$

solve (*sol*, *rhs*)

Compute $\text{sol} = UD^{-1}U^T x$

trace (*W=None*)

Compute the trace of A . If the weight W is given, compute the trace of $W^{1/2}AW^{1/2}$. This is equivalent to $\text{tr}_W(A) = \sum_i \lambda_i$, where λ_i are the generalized eigenvalues of $Ax = \lambda W^{-1}x$.

Note: If U is a W -orthogonal matrix then $\text{tr}_W(A) = \sum_i D(i, i)$.

trace2 (*W=None*)

Compute the trace of AA (Note this is the square of Frobenius norm, since A is symmetric). If the weight W is provided, it will compute the trace of $(AW)^2$.

This is equivalent to $\text{tr}_W(A) = \sum_i \lambda_i^2$, where λ_i are the generalized eigenvalues of $Ax = \lambda W^{-1}x$.

Note: If U is a W -orthogonal matrix then $\text{tr}_W(A) = \sum_i D(i, i)^2$.

hippylib.algorithms.multivector module

`hippylib.algorithms.multivector.MatMvMult` (A, x, y)

`hippylib.algorithms.multivector.MatMvTranspmult` (A, x, y)

class `hippylib.algorithms.multivector.MultiVector` (**args, **kwargs*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Borthogonalize (B)

Returns QR decomposition of self. Q and R satisfy the following relations in exact arithmetic

$$R = Z, \quad (1),$$

$$Q^*BQ = I, \quad (2),$$

$$Q^*BZ = R, \quad (3),$$

$$ZR^{-1} = Q, \quad (4).$$

Returns:

Bq of type `MultiVector` -> B^{-1} -orthogonal vectors r of type `ndarray` -> The r of the QR decomposition.

Note: `self` is overwritten by Q .

_exportFile ($Vh, filename, varname, normalize$)

Specialization of export using `dl.File`

_exportXDMF ($Vh, filename, varname, normalize$)

Specialization of export using `dl.File`

_mgs_reortho ()

_mgs_stable (B)

Returns QR decomposition of self, which satisfies conditions (1)–(4). Uses Modified Gram-Schmidt with re-orthogonalization (Rutishauser variant) for computing the B -orthogonal QR factorization.

References:

1. A.K. Saibaba, J. Lee and P.K. Kitanidis, *Randomized algorithms for Generalized Hermitian Eigenvalue Problems with application to computing Karhunen-Loe've expansion* <http://arxiv.org/abs/1307.6885>
2. W. Gander, *Algorithms for the QR decomposition*. Res. Rep, 80(02), 1980

<https://github.com/arvindks/kle>

dot_mv (mv)

dot_v (v)

export (*Vh, filename, varname='mv', normalize=False*)

Export in paraview this multivector.

Inputs:

- *Vh*: the parameter finite element space.
- *filename*: the name of the paraview output file.
- *varname*: the name of the paraview variable.
- *normalize*: if `True` the vector is rescaled such that $\|u\|_\infty = 1$.

orthogonalize ()

Returns QR decomposition of self. Q and R satisfy the following relations in exact arithmetic

$$QR = Z, \quad (1),$$

$$Q^*Q = I, \quad (2),$$

$$Q^*Z = R, \quad (3),$$

$$ZR^{-1} = Q, \quad (4).$$

Returns:

r of type `ndarray` -> The *r* of the QR decomposition

Note: `self` is overwritten by Q .

`hippylib.algorithms.multivector.MvDSmatMult` (*X, A, Y*)

hippylib.algorithms.randomizedEigensolver module

`hippylib.algorithms.randomizedEigensolver.check_g` (*A, B, U, d*)

Test the frobenious norm of $U^T BU - I_k$.

Test the frobenious norm of $(V^T AV) - I_k$, with $V = UD^{-1/2}$.

Test the l_2 norm of the residual: $r[i] = AU[i] - d[i]BU[i]$.

`hippylib.algorithms.randomizedEigensolver.check_std` (*A, U, d*)

Test the frobenious norm of $U^T U - I_k$.

Test the frobenious norm of $(V^T AV) - I_k$, with $V = UD^{-1/2}$.

Test the l_2 norm of the residual: $r[i] = AU[i] - d[i]U[i]$.

`hippylib.algorithms.randomizedEigensolver.doublePass` (*A, Omega, k, s, check=False*)

The double pass algorithm for the HEP as presented in [1].

Inputs:

- *A*: the operator for which we need to estimate the dominant eigenpairs.
- *Omega*: a random gaussian matrix with $m \geq k$ columns.
- *k*: the number of eigenpairs to extract.
- *s*: the number of power iterations for selecting the subspace.

Outputs:

- d: the estimate of the k dominant eigenvalues of A .
- U: the estimate of the k dominant eigenvectors of A , $U^T U = I_k$.

`hippylib.algorithms.randomizedEigensolver.doublePassG(A, B, Binv, Omega, k, s=1, check=False)`

The double pass algorithm for the GHEP as presented in [2].

Inputs:

- A: the operator for which we need to estimate the dominant generalized eigenpairs.
- B: the right-hand side operator.
- Binv: the inverse of the right-hand side operator.
- Omega: a random gaussian matrix with $m \geq k$ columns.
- k: the number of eigenpairs to extract.
- s: the number of power iterations for selecting the subspace.

Outputs:

- d: the estimate of the k dominant eigenvalues of A .
- U: the estimate of the k dominant eigenvectors of A , $U^T B U = I_k$.

`hippylib.algorithms.randomizedEigensolver.singlePass(A, Omega, k, s=1, check=False)`

The single pass algorithm for the Hermitian Eigenvalues Problems (HEP) as presented in [1].

Inputs:

- A: the operator for which we need to estimate the dominant eigenpairs.
- Omega: a random gaussian matrix with $m \geq k$ columns.
- k: the number of eigenpairs to extract.

Outputs:

- d: the estimate of the k dominant eigenvalues of A .
- U: the estimate of the k dominant eigenvectors of A , $U^T U = I_k$.

`hippylib.algorithms.randomizedEigensolver.singlePassG(A, B, Binv, Omega, k, s=1, check=False)`

The single pass algorithm for the Generalized Hermitian Eigenvalues Problems (GHEP) as presented in [2].

Inputs:

- A: the operator for which we need to estimate the dominant generalized eigenpairs.
- B: the right-hand side operator.
- Binv: the inverse of the right-hand side operator.
- Omega: a random gaussian matrix with $m \geq k$ columns.
- k: the number of eigenpairs to extract.
- s: the number of power iterations for selecting the subspace.

Outputs:

- d: the estimate of the k dominant eigenvalues of A .
- U: the estimate of the k dominant eigenvectors of A , $U^T B U = I_k$.

hippylib.algorithms.randomizedSVD module

`hippylib.algorithms.randomizedSVD.accuracyEnhancedSVD(A, Omega, k, s=1, check=False)`

The accuracy enhanced randomized singular value decomposition from [2].

Inputs:

- A: the $m \times n$ rectangular operator for which we need to estimate the dominant left-right singular vector / value triplets.
- Omega: a random gaussian matrix with $m \geq k$ columns.
- k: the number of eigenpairs to extract.

Outputs: - U: the estimate of the k dominant left singular vectors of A , $U^T U = I_k$. - sigma: the estimate of the k dominant singular values of A . - V: the estimate of the k dominant right singular vectors of A , $V^T V = I_k$.

`hippylib.algorithms.randomizedSVD.check_SVD(A, U, sigma, V, tol=0.1)`

Test the frobenious norm of $U^T U - I_k$.

Test the frobenious norm of $(V^T V) - I_k$.

Test the l_2 norm of the residual: $r_1[i] = U[i]^T A V[i] - \text{sigma}[i]$.

Test the l_2 norm of the residual: $r_2[i] = V[i]^T A^T U[i] - \text{sigma}[i]$.

`hippylib.algorithms.randomizedSVD.singlePassSVD(A, Omega_c, Omega_r, k, check=False)`

The single pass randomized singular value decomposition from [2].

Inputs:

- A: the $m \times n$ rectangular operator for which we need to estimate the dominant left-right singular vector / value triplets.
- Omega_c: an $n \times (k + p)$ random gaussian matrix with $n \geq k$ columns.
- Omega_r: an $m \times (k + p)$ random gaussian matrix with $m \geq k$ columns.
- k: the number of eigenpairs to extract.

Outputs: - U: the estimate of the k dominant left singular vectors of A , $U^T U = I_k$. - sigma: the estimate of the k dominant singular values of A . - V: the estimate of the k dominant right singular vectors of A , $V^T V = I_k$.

hippylib.algorithms.steepestDescent module

`class hippylib.algorithms.steepestDescent.SteepestDescent(model, parameters=<hippylib.utils.parameterList.ParameterList object>)`

Bases: `object`

Prior-preconditioned Steepest Descent to solve constrained optimization problems in the reduced parameter space. Globalization is performed using the Armijo sufficient reduction condition (backtracking). The stopping criterion is based on a control on the norm of the gradient.

The user must provide a model that describes the forward problem, cost functionals, and all the derivatives for the gradient.

More specifically the model object should implement following methods:

- `generate_vector()` -> generate the object containing state, parameter, adjoint.
- `cost(x)` -> evaluate the cost functional, report regularization part and misfit separately.

- `solveFwd(out, x)` -> solve the possibly non linear forward problem.
- `solveAdj(out, x)` -> solve the linear adjoint problem.
- `evalGradientParameter(x, out)` -> evaluate the gradient of the parameter and compute its norm.
- `Rsolver()` -> A solver for the regularization term.

Type `help(Model)` for additional information.

Initialize the Steepest Descent solver. Type `SteepestDescent_ParameterList().showMe()` for list of default parameters and their descriptions.

solve (*x*)

Solve the constrained optimization problem with initial guess $x = [u, a, p]$. Return the solution $[u, a, p]$.

Note: x will be overwritten.

```

    termination_reasons = ['Maximum number of Iteration reached', 'Norm of the gradient le
hippylib.algorithms.steepestDescent.SteepestDescent_ParameterList()
```

hippylib.algorithms.traceEstimator module

```

class hippylib.algorithms.traceEstimator.TraceEstimator(A, solve_mode=False,
                                                         accuracy=0.1,
                                                         init_vector=None, ran-
                                                         dom_engine=<function
                                                         rademacher_engine>,
                                                         mpi_comm=<sphinx.ext.autodoc.importer._MockO
                                                         object>)
```

Bases: `object`

An unbiased stochastic estimator for the trace of A , $d = \sum_{j=1}^k (v_j, Av_j)$, where

- v_j are i.i.d. Rademacher or Gaussian random vectors.
- (\cdot, \cdot) represents the inner product.

The number of samples k is estimated at run time based on the variance of the estimator.

Reference: Haim Avron and Sivan Toledo, Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix, Journal of the ACM (JACM), 58 (2011), p. 17.

Constructor:

- A : an operator
- `solve_mode`: if `True` we estimate the trace of A^{-1} , otherwise of A .
- **code:accuracy:** we stop when the standard deviation of the estimator is less then `accuracy`*tr(:code:`A)`.
- **init_vector:** use a custom function to initialize a vector compatible with the range/domain of A .
- `random_engine`: which type of i.i.d. random variables to use (Rademacher or Gaussian).

```

hippylib.algorithms.traceEstimator.gaussian_engine(v)
Generate a vector of  $n$  i.i.d. standard normal variables.
```

`hippylib.algorithms.traceEstimator.rademacher_engine(v)`
 Generate a vector of n i.i.d. Rademacher variables.

Module contents

hippylib.forward_uq package

Submodules

hippylib.forward_uq.qoi module

class `hippylib.forward_uq.qoi.Qoi`

Bases: `object`

Abstract class to model the Quantity of Interest. In the following x will denote the variable $[u, m, p]$, denoting respectively the state u , the parameter m , and the adjoint variable p .

The methods in the class QOI will usually access the state u and possibly the parameter m .

apply_ij (i, j, dir, out)

Apply the second variation δ_{ij} ($i, j = \text{STATE}, \text{PARAMETER}$) of the cost in direction dir .

eval (x)

Given x evaluate the cost functional. Only the state u and (possibly) the parameter m are accessed.

grad (i, x, g)

Evaluate the gradient with respect to the state. Only the state u and (possibly) the parameter m are accessed.

setLinearizationPoint (x)

`hippylib.forward_uq.qoi.qoiVerify(qoi, x, generate_state, h=None, plotting=True)`

`hippylib.forward_uq.qoi.qoiVerifyPlotErrors(eps, err_grad, err_H)`

hippylib.forward_uq.parameter2QoiMap module

class `hippylib.forward_uq.parameter2QoiMap.Parameter2QoiHessian(p2qoimap)`

Bases: `object`

This class implements matrix free application of the reduced hessian operator.

The constructor takes the following parameters:

- `p2qoimap` - the object that describes the parameter-to-qoi map

Construct the Hessian Operator of the parameter-to-qoi map

init_vector (x, dim)

Reshape the Vector x so that it is compatible with the reduced Hessian operator.

Parameters:

- x - the vector to reshape
- dim - if 0 then x will be reshaped to be compatible with the range of the reduced Hessian, if 1 then x will be reshaped to be compatible with the domain of the reduced Hessian

Note: Since the reduced Hessian is a self adjoint operator, the range and the domain is the same. Either way, we chose to add the parameter `dim` for consistency with the interface of `dolfin.Matrix`.

inner (*x*, *y*)

Perform the inner product between *x* and *y* in the norm induced by the Hessian *H*, i.e. $(x, y)_H = x^T H y$

mult (*x*, *y*)

Apply the Hessian of the parameter-to-qoi map to the vector *x* Return the result in *y*.

class hippylib.forward_uq.parameter2QoiMap.**Parameter2QoiMap** (*problem*, *qoi*)

Bases: object

Create a parameter-to-qoi map given:

- *problem* - the description of the forward/adjoint problem and all the sensitivities
- *qoi* - the quantity of interest as a function of the state and parameter

applyC (*dm*, *out*)

Apply the *C* block of the Hessian to a (incremental) parameter variable. *out* = $C dm$

Parameters:

- *dm* the (incremental) parameter variable
- *out* the action of the *C* block on `dm`

Note: this routine assumes that *out* has the correct shape.

applyCt (*dp*, *out*)

Apply the transpose of the *C* block of the Hessian to a (incremental) adjoint variable. *out* = $C^T dp$

Parameters:

- *dp* the (incremental) adjoint variable
- *out* the action of the C^T block on *dp*

Note: this routine assumes that *out* has the correct shape.

applyWmm (*dm*, *out*)

Apply the W_{mm} block of the Hessian to a (incremental) parameter variable. *out* = $W_{mm} dm$

Parameters:

- *dm* the (incremental) parameter variable
- *out* the action of W_{mm} on *dm*

Note: this routine assumes that *out* has the correct shape.

applyWmu (*du*, *out*)

Apply the W_{mu} block of the Hessian to a (incremental) state variable. *out* = $W_{mu} du$

Parameters:

- *du* the (incremental) state variable

- `out` the action of the W_{mu} block on `du`

Note: this routine assumes that `out` has the correct shape.

applyWum (*dm, out*)

Apply the W_{um} block of the Hessian to a (incremental) parameter variable. $out = W_{um}dm$

Parameters:

- `dm` the (incremental) parameter variable
- `out` the action of the W_{um} block on `du`

Note: this routine assumes that `out` has the correct shape.

applyWuu (*du, out*)

Apply the W_{uu} block of the Hessian to a (incremental) state variable. $out = W_{uu}du$

Parameters:

- `du` the (incremental) state variable
- `out` the action of the W_{uu} block on `du`

Note: this routine assumes that `out` has the correct shape.

eval (*x*)

Given the list $x = [u, m, p]$ which describes the state, parameter, and adjoint variable compute the QOI.

Note: `p` is not needed to compute the QOI

evalGradientParameter (*x, mg*)

Evaluate the gradient for the variational parameter equation at the point $x = [u, m, p]$.

Parameters:

- $x = [u, m, p]$ the point at which to evaluate the gradient.
- `mg` the variational gradient ($g, mtest$) being `mtest` a test function in the parameter space (Output parameter)

generate_vector (*component='ALL'*)

By default, return the list $[u, m, p]$ where:

- `u` is any object that describes the state variable
- `m` is a `dolfin.Vector` object that describes the parameter variable. (Need to support linear algebra operations)
- `p` is any object that describes the adjoint variable

If `component == STATE` return only `u`

If `component == PARAMETER` return only `m`

If `component == ADJOINT` return only `p`

hessian (*m=None, x=None*)

Evaluate the Hessian of parameter-to-qoi map.

If a realization of the parameter *m* is given, this function will automatically compute the state *u* and adjoint *p*.

As an alternative one can provide directly $x = [u, m, p]$

Returns an object of type `ReducedHessianQOI` which provides the Hessian-apply functionality

init_parameter (*m*)

Reshape *m* so that it is compatible with the parameter variable

reduced_eval (*m*)

Evaluate the parameter-to-qoi map at a given realization *m*

Note: This evaluation requires the solution of a forward solve

reduced_gradient (*m, g*)

Evaluate the gradient of parameter-to-qoi map at a given realization *m*

Note: This evaluation requires the solution of a forward and adjoint solve

setLinearizationPoint (*x*)

Specify the point $x = [u, m, p]$ at which the Hessian operator needs to be evaluated.

Parameters:

- $x = [u, m, p]$: the point at which the Hessian needs to be evaluated.

solveAdj (*out, x*)

Solve the linear adjoint problem.

Parameters:

- *out* - is the solution of the adjoint problem (i.e. the adjoint *p*) (Output parameter)
- $x = [u, m, p]$ provides
 - 1) the parameter variable *m* for assembling the adjoint operator
 - 2) the state variable *u* for assembling the adjoint right hand side

Note: *p* is not accessed

solveAdjIncremental (*sol, rhs*)

Solve the incremental adjoint problem for a given *rhs*

Parameters:

- *sol* the solution of the incremental adjoint problem (Output)
- *rhs* the right hand side of the linear system

solveFwd (*out, x*)

Solve the (possibly non-linear) forward problem.

Parameters:

- *out* - is the solution of the forward problem (i.e. the state) (Output parameters)

- $x = [u, m, p]$ provides
 - 1) the parameter variable m for the solution of the forward problem
 - 2) the initial guess u if the forward problem is non-linear

Note: p is not accessed

solveFwdIncremental (*sol, rhs*)

Solve the linearized (incremental) forward problem for a given rhs

Parameters:

- *sol* the solution of the linearized forward problem (Output)
- *rhs* the right hand side of the linear system

hippylib.forward_uq.parameter2QoiMap.**parameter2QoiMapVerify** (*rQOI, m0, h=None, eps=None, plotting=True, verbose=True*)

Verify the gradient and the Hessian of a parameter-to-qoi map. It will produce two loglog plots of the finite difference checks for the gradient and for the Hessian. It will also check for symmetry of the Hessian.

hippylib.forward_uq.parameter2QoiMap.**parameter2QoiMapVerifyPlotErrors** (*eps, err_grad, err_H*)

hippylib.forward_uq.taylorApproximationQoi module

class hippylib.forward_uq.taylorApproximationQoi.**TaylorApproximationQoi** (*p2qoimap, distribution*)

Bases: object

This class computes the first and second order Taylor approximation of the parameter-to-qoi map. It provides methods to evaluate the Taylor approximation for a specific realization of the parameter and to analytically compute Expectation and Variance of the Taylor approximation with respect to a Gaussian propability distribution.

Constructor:

- *p2qoimap* - an object of type `ReducedQOI` that describes the parameter-to-qoi map
- *distribution* - an object of type `hippylib._Prior` that describes the prior Gaussian distribution.

computeLowRankFactorization (*Omega*)

Compute the LowRank Factorization of the prior-preconditioned Hessian of the parameter-to-qoi map.

The trace of the the prior-preconditioned Hessian is also computed as a post-process of the LowRank factorization.

This method needs to be called before trying to compute the moments of the the quadratic Taylor approx of the parameter-to-qoi map.

Inputs:

- *Omega* - Gaussian random matrix for randomized method

- `k` - the number of eigenpairs to be retained in the LowRank factorization of the prior-preconditioned Hessian.

eval (*m*, *order*=2)

Evaluates the Taylor approx of the qoi for a given realization of the parameter.

Input:

- `m` - a specific realization of the uncertain parameter
- `order` - is the order of the Taylor approximation, currently 1 (linear) or 2 (quadratic)

expectedValue (*order*=2)

Returns the expected value (computed analytically) of the qoi with respect to a Gaussian distribution for the parameter.

Input:

- `order` - is the order of the Taylor approximation, currently 1 (linear) or 2 (quadratic)

variance (*order*=2)

Returns the variance (computed analytically) of the qoi with respect to a Gaussian distribution for the parameter.

Input:

- `order` - is the order of the Taylor approximation, currently 1 (linear) or 2 (quadratic)

`hippylib.forward_uq.taylorApproximationQoi.plotEigenvalues` (*d*)

Plots the eigenvalues *d* in a semilogy scale. Positive eigenvalues are marked in blue, negative eigenvalues are marked in red.

hippylib.forward_uq.varianceReductionMC module

`hippylib.forward_uq.varianceReductionMC.varianceReductionMC` (*prior*, *rqoi*,
taylor_qoi,
nsamples, *file-*
name='realizations.txt')

This function computes Monte Carlo Estimates for forward propagation of uncertainty. The uncertain parameter satisfies a Gaussian distribution with known mean and covariance (describes as the inverse of a differential operator). Convergence of the Monte Carlo estimates is accelerated using a variance reduction technique based on a Taylor approximation of the parameter-to-qoi map.

Inputs:

- `prior` - an object of type `:code:hippylib._Prior` that allows to generate samples from the prior distribution
- `rqoi` - an object of type `ReducedQOI` that describes the parameter-to-qoi map
- `taylor_qoi` - an object of type `TaylorApproximationQOI` that computes the first and second order Taylor approximation of the qoi
- `nsamples` - an integer representing the number of samples for the MC estimates
- `filename` - a string containing the name of the file where the computed qoi and its Taylor approximations (for each realization of the parameter) are saved

Outputs:

- Sample mean of the quantity of interest *q*, its Taylor approx *q1* and *q2*, and corrections *y1*=*q-q1* and *y2*=*q-q2*.

- MSE (Mean square error) of the standard MC, and the variance reduced MC using $q1$ and $q2$.

Note: The variate control MC estimator can be computed off-line by postprocessing the file containing the values of the computed qoi and Taylor approximations.

Module contents

hippylib.mcmc package

Submodules

hippylib.mcmc.chain module

```
class hippylib.mcmc.chain.MCMC(kernel)
    Bases: object
        consume_random()
        run(m0, qoi=None, tracer=None)
class hippylib.mcmc.chain.NullQoi
    Bases: object
        eval(x)
class hippylib.mcmc.chain.SampleStruct(kernel)
    Bases: object
        assign(other)
```

hippylib.mcmc.diagnostics module

```
hippylib.mcmc.diagnostics._acorr(mean_free_samples, lag, norm=1)
hippylib.mcmc.diagnostics._acorr_vs_lag(samples, max_lag)
hippylib.mcmc.diagnostics.integratedAutocorrelationTime(samples, max_lag=None)
```

hippylib.mcmc.kernels module

```
class hippylib.mcmc.kernels.ISKernel(model, nu)
    Bases: object
        consume_random()
        delta(sample)
        derivativeInfo()
        init_sample(current)
        name()
        proposal(current)
        sample(current, proposed)
```

```
class hippylib.mcmc.kernels.MALAKernel (model)
```

```
    Bases: object
```

```
    acceptance_ratio (origin, destination)
```

```
    consume_random ()
```

```
    derivativeInfo ()
```

```
    init_sample (s)
```

```
    name ()
```

```
    proposal (current)
```

```
    sample (current, proposed)
```

```
class hippylib.mcmc.kernels.gpCNKernel (model, nu)
```

```
    Bases: object
```

Reference: F. J. PINSKI, G. SIMPOSN, A. STUART, H. WEBER, *Algorithms for Kullback-Leibler Approximation of Probability Measures in Infinite Dimensions*, <http://arxiv.org/pdf/1408.1920v1.pdf>, Alg. 5.2

```
    consume_random ()
```

```
    delta (sample)
```

```
    derivativeInfo ()
```

```
    init_sample (current)
```

```
    name ()
```

```
    proposal (current)
```

```
    sample (current, proposed)
```

```
class hippylib.mcmc.kernels.pCNKernel (model)
```

```
    Bases: object
```

```
    consume_random ()
```

```
    derivativeInfo ()
```

```
    init_sample (current)
```

```
    name ()
```

```
    proposal (current)
```

```
    sample (current, proposed)
```

hippylib.mcmc.tracers module

```
class hippylib.mcmc.tracers.FullTracer (n, Vh, par_fid=None, state_fid=None)
```

```
    Bases: object
```

```
    append (current, q)
```

```
class hippylib.mcmc.tracers.NullTracer
```

```
    Bases: object
```

```
    append (current, q)
```

```
class hippylib.mcmc.tracers.QoiTracer (n)
```

```
    Bases: object
```

append (*current*, *q*)

Module contents

hippylib.scheduling package

Submodules

hippylib.scheduling.collective module

class hippylib.scheduling.collective.**MultipleSerialPDEsCollective** (*comm*)

Bases: object

Parallel reduction utilities when several serial systems of PDEs (one per process) are solved concurrently.

comm is `mpi4py.MPI comm`

allReduce (*v*, *op*)

Case handled: - *v* is a scalar (`float`, `int`); - *v* is a numpy array (NOTE: *v* will be overwritten) - *v* is a `dolfin.Vector` (NOTE: *v* will be overwritten) Operation: *op* = "Sum" or "Avg" (case insensitive).

rank ()

size ()

class hippylib.scheduling.collective.**NullCollective**

Bases: object

No-overhead "Parallel" reduction utilities when a serial system of PDEs is solved on 1 process.

allReduce (*v*, *op*)

rank ()

size ()

Module contents

hippylib.utils package

Submodules

hippylib.utils.checkDolfinVersion module

hippylib.utils.checkDolfinVersion.**checkdlversion** ()

Check if FEniCS version is supported. Currently hIPPYlib requires FEniCS version 1.6.0 and newer.

hippylib.utils.checkDolfinVersion.**dlversion** ()

hippylib.utils.parameterList module

class hippylib.utils.parameterList.**ParameterList** (*data*)

Bases: object

A small abstract class for storing parameters and their description. This class will raise an exception if the key one tries to access is not present.

data is a dictionary where each value is the pair (value, description)

showMe (*indent=""*)

hippylib.utils.vector2function module

hippylib.utils.vector2function.**vector2Function** (*x, Vh, **kwargs*)

Wrap a finite element vector *x* into a finite element function in the space *Vh*. *kwargs* is optional keywords arguments to be passed to the construction of a dolfin *Function*.

hippylib.utils.random module

class hippylib.utils.random.**Random** (*myid=0, nproc=1, blocksize=1000000, seed=1*)

Bases: sphinx.ext.autodoc.importer._MockObject

This class handles parallel generation of random numbers in hippylib.

Create a parallel random number number generator.

INPUTS:

- *myid*: id of the calling process.
- *nproc*: number of processor in the communicator.
- *blocksize*: number of consecutive random number to be generated before jumping headed in the stream.
- *seed*: random seed to initialize the random engine.

normal (*sigma, out=None*)

Sample from normal distribution with given variance.

normal_perturb (*sigma, out*)

Add a normal perturbation to a Vector/MultiVector.

rademacher (*out=None*)

Sample from Rademacher distribution.

speckle (*M, out*)

Add a normal perturbation to a Vector/MultiVector.

uniform (*a, b, out=None*)

Sample from uniform distribution.

hippylib.utils.random.**parRandom**

This class handles parallel generation of random numbers in hippylib.

hippylib.utils.nb module

hippylib.utils.nb.**_mesh2triang** (*mesh*)

hippylib.utils.nb.**_mplot_cellfunction** (*cellfn*)

hippylib.utils.nb.**_mplot_function** (*f, vmin, vmax, logscale*)

hippylib.utils.nb.**animate** (*Vh, state, same_colorbar=True, colorbar=True, subplot_loc=None, mytitle=None, show_axis='off', logscale=False*)

Show animation for a :code:TimeDependentVector

```
hippylib.utils.nb.coarsen_v (fun, nx=16, ny=16)
hippylib.utils.nb.multil_plot (objs, titles, same_colorbar=True, show_axis='off',
                               logscale=False, vmin=None, vmax=None, cmap=None)
    Plot a list of generic dolfin object in a single row
hippylib.utils.nb.plot (obj, colorbar=True, subplot_loc=None, mytitle=None, show_axis='off',
                        vmin=None, vmax=None, logscale=False, cmap=None)
    Plot a generic dolfin object (if supported)
hippylib.utils.nb.plot_eigenvalues (d, mytitle=None, subplot_loc=None)
    Plot eigenvalues
hippylib.utils.nb.plot_eigenvectors (Vh, U, mytitle, which=[0, 1, 2, 5, 10, 15], cmap=None)
    Plot specified vectors in a :code:MultiVector
hippylib.utils.nb.plot_pts (points, values, colorbar=True, subplot_loc=None, mytitle=None,
                            show_axis='on', vmin=None, vmax=None, xlim=(0, 1), ylim=(0, 1),
                            cmap=None)
    Plot a cloud of points
hippylib.utils.nb.show_solution (Vh, ic, state, same_colorbar=True, colorbar=True, myti-
                                tle=None, show_axis='off', logscale=False, times=[0, 0.4, 1.0,
                                2.0, 3.0, 4.0], cmap=None)
    Plot a :code:TimeDependentVector at specified time steps
```

Module contents

4.1.2 Module contents

hippylib implements state-of-the-art scalable algorithms for PDE-based deterministic and Bayesian inverse problems. It builds on FEniCS (<http://fenicsproject.org/>) (a parallel finite element library) for the discretization of the PDE and on PETSc (<http://www.mcs.anl.gov/petsc/>) for scalable and efficient linear algebra operations and solvers.

For building instructions, see the file INSTALL. Copyright information and licensing restrictions can be found in the file COPYRIGHT.

The best starting point for new users interested in hippylib's features are the interactive tutorials in the notebooks folder.

Conceptually, hippylib can be viewed as a toolbox that provides the building blocks for experimenting new ideas and developing scalable algorithms for PDE-based deterministic and Bayesian inverse problems.

h

`hippylib`, 57
`hippylib.algorithms`, 47
`hippylib.algorithms.bfgs`, 37
`hippylib.algorithms.cgsampler`, 38
`hippylib.algorithms.cgsolverSteihaug`, 39
`hippylib.algorithms.linalg`, 40
`hippylib.algorithms.lowRankOperator`, 41
`hippylib.algorithms.multivector`, 42
`hippylib.algorithms.NewtonCG`, 35
`hippylib.algorithms.randomizedEigensolver`, 43
`hippylib.algorithms.randomizedSVD`, 45
`hippylib.algorithms.steepestDescent`, 45
`hippylib.algorithms.traceEstimator`, 46
`hippylib.forward_uq`, 53
`hippylib.forward_uq.parameter2QoiMap`, 47
`hippylib.forward_uq.qoi`, 47
`hippylib.forward_uq.taylorApproximationQoi`, 51
`hippylib.forward_uq.varianceReductionMC`, 52
`hippylib.mcmc`, 55
`hippylib.mcmc.chain`, 53
`hippylib.mcmc.diagnostics`, 53
`hippylib.mcmc.kernels`, 53
`hippylib.mcmc.tracers`, 54
`hippylib.modeling`, 35
`hippylib.modeling.expression`, 21
`hippylib.modeling.misfit`, 21
`hippylib.modeling.model`, 24
`hippylib.modeling.modelVerify`, 27
`hippylib.modeling.PDEProblem`, 19
`hippylib.modeling.pointwiseObservation`, 27
`hippylib.modeling.posterior`, 28
`hippylib.modeling.prior`, 30
`hippylib.modeling.reducedHessian`, 33
`hippylib.modeling.timeDependentVector`, 35
`hippylib.modeling.variables`, 35
`hippylib.scheduling`, 55
`hippylib.scheduling.collective`, 55
`hippylib.utils`, 57
`hippylib.utils.checkDolfinVersion`, 55
`hippylib.utils.nb`, 56
`hippylib.utils.parameterList`, 55
`hippylib.utils.random`, 56
`hippylib.utils.vector2function`, 56

Symbols

[_BilaplacianR](#) (class in `hippylib.modeling.prior`), 32
[_BilaplacianRsolver](#) (class in `hippylib.modeling.prior`), 32
[_Prior](#) (class in `hippylib.modeling.prior`), 32
[_RinvM](#) (class in `hippylib.modeling.prior`), 33
[_acorr\(\)](#) (in module `hippylib.mcmc.diagnostics`), 53
[_acorr_vs_lag\(\)](#) (in module `hippylib.mcmc.diagnostics`), 53
[_createLUSolver\(\)](#) (`hippylib.modeling.PDEProblem.PDEVariationalProblem` method), 20
[_exportFile\(\)](#) (`hippylib.algorithms.multivector.MultiVector` method), 42
[_exportXDMF\(\)](#) (`hippylib.algorithms.multivector.MultiVector` method), 42
[_mesh2triang\(\)](#) (in module `hippylib.utils.nb`), 56
[_mgs_reortho\(\)](#) (`hippylib.algorithms.multivector.MultiVector` method), 42
[_mgs_stable\(\)](#) (`hippylib.algorithms.multivector.MultiVector` method), 42
[_mplot_cellfunction\(\)](#) (in module `hippylib.utils.nb`), 56
[_mplot_function\(\)](#) (in module `hippylib.utils.nb`), 56
[_sample_given_prior\(\)](#) (`hippylib.modeling.posterior.GaussianLRPosterior` method), 28
[_sample_given_white_noise\(\)](#) (`hippylib.modeling.posterior.GaussianLRPosterior` method), 28
[_solve_ls\(\)](#) (`hippylib.algorithms.NewtonCG.ReducedSpaceNewtonCG` method), 36
[_solve_tr\(\)](#) (`hippylib.algorithms.NewtonCG.ReducedSpaceNewtonCG` method), 37

A

[acceptance_ratio\(\)](#) (`hippylib.mcmc.kernels.MALAKernel` method), 54
[accuracyEnhancedSVD\(\)](#) (in module `hippylib.algorithms.randomizedSVD`), 45
[allReduce\(\)](#) (`hippylib.scheduling.collective.MultipleSerialPDEsCollective` method), 55
[allReduce\(\)](#) (`hippylib.scheduling.collective.NullCollective` method), 55
[animate\(\)](#) (in module `hippylib.utils.nb`), 56
[append\(\)](#) (`hippylib.mcmc.tracers.FullTracer` method), 54
[append\(\)](#) (`hippylib.mcmc.tracers.NullTracer` method), 54
[append\(\)](#) (`hippylib.mcmc.tracers.QoiTracer` method), 54
[append\(\)](#) (`hippylib.modeling.misfit.MultiStateMisfit` method), 23
[apply_ij\(\)](#) (`hippylib.forward_uq.qoi.Qoi` method), 47
[apply_ij\(\)](#) (`hippylib.modeling.misfit.ContinuousStateObservation` method), 21
[apply_ij\(\)](#) (`hippylib.modeling.misfit.Misfit` method), 22
[apply_ij\(\)](#) (`hippylib.modeling.misfit.MultiStateMisfit` method), 23
[apply_ij\(\)](#) (`hippylib.modeling.misfit.MultiPointwiseStateObservation` method), 22
[apply_ij\(\)](#) (`hippylib.modeling.misfit.PointwiseStateObservation` method), 23
[apply_ij\(\)](#) (`hippylib.modeling.model.Model` method), 25
[apply_ij\(\)](#) (`hippylib.modeling.PDEProblem.PDEProblem` method), 19
[apply_ij\(\)](#) (`hippylib.modeling.PDEProblem.PDEVariationalProblem` method), 20
[apply_ijk\(\)](#) (`hippylib.modeling.PDEProblem.PDEProblem` method), 20

`method`), 19
`apply_ijk()` (`hippylib.modeling.PDEProblem.PDEVariationalProblem` `method`), 20
`applyC()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 48
`applyC()` (`hippylib.modeling.model.Model` `method`), 24
`applyCt()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 48
`applyCt()` (`hippylib.modeling.model.Model` `method`), 24
`applyR()` (`hippylib.modeling.model.Model` `method`), 24
`applyWmm()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 48
`applyWmm()` (`hippylib.modeling.model.Model` `method`), 25
`applyWmu()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 48
`applyWmu()` (`hippylib.modeling.model.Model` `method`), 25
`applyWum()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 49
`applyWum()` (`hippylib.modeling.model.Model` `method`), 25
`applyWuu()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 49
`applyWuu()` (`hippylib.modeling.model.Model` `method`), 25
`assemblePointwiseObservation()` (in module `hippylib.modeling.pointwiseObservation`), 27
`assign()` (`hippylib.mcmc.chain.SampleStruct` `method`), 53
`axpy()` (`hippylib.modeling.timeDependentVector.TimeDependentVector` `method`), 35

B

`BFGS` (class in `hippylib.algorithms.bfgs`), 37
`BFGS_operator` (class in `hippylib.algorithms.bfgs`), 38
`BFGS_ParameterList()` (in module `hippylib.algorithms.bfgs`), 38
`BFGSoperator_ParameterList()` (in module `hippylib.algorithms.bfgs`), 38
`BiLaplacianComputeCoefficients()` (in module `hippylib.modeling.prior`), 30
`BiLaplacianPrior()` (in module `hippylib.modeling.prior`), 30
`Borthogonalize()` (`hippylib.algorithms.multivector.MultiVector` `method`), 42

C

`CGSampler` (class in `hippylib.algorithms.cgsampler`), 38
`CGSolverSteihaug` (class in `hippylib.algorithms.cgsolverSteihaug`), 39
`CGSolverSteihaug_ParameterList()` (in module `hippylib.algorithms.cgsolverSteihaug`), 40
`check_g()` (in module `hippylib.algorithms.randomizedEigensolver`), 43
`check_std()` (in module `hippylib.algorithms.randomizedEigensolver`), 43
`check_SVD()` (in module `hippylib.algorithms.randomizedSVD`), 45
`checkdlversion()` (in module `hippylib.utils.checkDolfinVersion`), 55
`coarsen_v()` (in module `hippylib.utils.nb`), 56
`computeRankFactorization()` (`hippylib.forward_uq.taylorApproximationQoi.TaylorApproximationQoi` `method`), 51
`consume_random()` (`hippylib.mcmc.chain.MCMC` `method`), 53
`consume_random()` (`hippylib.mcmc.kernels.gpCNKernel` `method`), 54
`consume_random()` (`hippylib.mcmc.kernels.ISKernel` `method`), 53
`consume_random()` (`hippylib.mcmc.kernels.MALAKernel` `method`), 54
`consume_random()` (`hippylib.mcmc.kernels.pCNKernel` `method`), 54
`ContinuousStateObservation` (class in `hippylib.modeling.misfit`), 21
`copy()` (`hippylib.modeling.timeDependentVector.TimeDependentVector` `method`), 35
`cost()` (`hippylib.modeling.misfit.ContinuousStateObservation` `method`), 21
`cost()` (`hippylib.modeling.misfit.Misfit` `method`), 22
`cost()` (`hippylib.modeling.misfit.MultiStateMisfit` `method`), 23
`cost()` (`hippylib.modeling.misfit.MultiPointwiseStateObservation` `method`), 22
`cost()` (`hippylib.modeling.misfit.PointwiseStateObservation` `method`), 23
`cost()` (`hippylib.modeling.model.Model` `method`), 25
`cost()` (`hippylib.modeling.posterior.GaussianLRPosterior` `method`), 28
`cost()` (`hippylib.modeling.prior._Prior` `method`), 33

D

`delta()` (`hippylib.mcmc.kernels.gpCNKernel` `method`), 54
`delta()` (`hippylib.mcmc.kernels.ISKernel` `method`), 53

`derivativeInfo()` (hip-
`pylib.mcmc.kernels.gpCNKernel` method),
 54
`derivativeInfo()` (hippylib.mcmc.kernels.ISKernel
 method), 53
`derivativeInfo()` (hip-
`pylib.mcmc.kernels.MALAKernel` method),
 54
`derivativeInfo()` (hip-
`pylib.mcmc.kernels.pCNKernel` method),
 54
`DiagonalOperator` (class in hip-
`pylib.algorithms.linalg`), 40
`dlversion()` (in module hip-
`pylib.utils.checkDolfinVersion`), 55
`dot_mv()` (hippylib.algorithms.multivector.MultiVector
 method), 42
`dot_v()` (hippylib.algorithms.multivector.MultiVector
 method), 42
`doublePass()` (in module hip-
`pylib.algorithms.randomizedEigensolver`),
 43
`doublePassG()` (in module hip-
`pylib.algorithms.randomizedEigensolver`),
 44

E

`estimate_diagonal_inv2()` (in module hip-
`pylib.algorithms.linalg`), 41
`eval()` (hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap
 method), 49
`eval()` (hippylib.forward_uq.qoi.Qoi method), 47
`eval()` (hippylib.forward_uq.taylorApproximationQoi.TaylorApproximationQoi
 method), 52
`eval()` (hippylib.mcmc.chain.NullQoi method), 53
`evalGradientParameter()` (hip-
`pylib.forward_uq.parameter2QoiMap.Parameter2QoiMap`
 method), 49
`evalGradientParameter()` (hip-
`pylib.modeling.model.Model` method), 26
`evalGradientParameter()` (hip-
`pylib.modeling.PDEProblem.PDEProblem`
 method), 19
`evalGradientParameter()` (hip-
`pylib.modeling.PDEProblem.PDEVariationalProblem`
 method), 20
`expectedValue()` (hip-
`pylib.forward_uq.taylorApproximationQoi.TaylorApproximationQoi`
 method), 52
`export()` (hippylib.algorithms.multivector.MultiVector
 method), 42
`exportPointwiseObservation()` (in module
`hippylib.modeling.pointwiseObservation`), 28

F

`FDHessian` (class in hip-
`pylib.modeling.reducedHessian`), 33
`FullTracer` (class in hippylib.mcmc.tracers), 54

G

`gaussian_engine()` (in module hip-
`pylib.algorithms.traceEstimator`), 46
`GaussianLRPosterior` (class in hip-
`pylib.modeling.posterior`), 28
`GaussianRealPrior` (class in hip-
`pylib.modeling.prior`), 30
`generate_parameter()` (hip-
`pylib.modeling.PDEProblem.PDEProblem`
 method), 19
`generate_parameter()` (hip-
`pylib.modeling.PDEProblem.PDEVariationalProblem`
 method), 20
`generate_state()` (hip-
`pylib.modeling.PDEProblem.PDEProblem`
 method), 19
`generate_state()` (hip-
`pylib.modeling.PDEProblem.PDEVariationalProblem`
 method), 20
`generate_vector()` (hip-
`pylib.forward_uq.parameter2QoiMap.Parameter2QoiMap`
 method), 49
`generate_vector()` (hip-
`pylib.modeling.model.Model` method), 26
`get_diagonal()` (hip-
`pylib.algorithms.lowRankOperator.LowRankOperator`
 method), 41
`get_diagonal()` (in module hip-
`pylib.algorithms.linalg`), 41
`GetFromOwnedGid()` (in module hip-
`pylib.algorithms.linalg`), 40
`getHessianPreconditioner()` (hip-
`pylib.modeling.prior._Prior` method), 33
`GNHessian()` (hippylib.modeling.reducedHessian.ReducedHessian
 method), 34
`gpCNKernel` (class in hippylib.mcmc.kernels), 54
`grad()` (hippylib.forward_uq.qoi.Qoi method), 47
`grad()` (hippylib.modeling.misfit.ContinuousStateObservation
 method), 22
`grad()` (hippylib.modeling.misfit.Misfit method), 22
`grad()` (hippylib.modeling.misfit.MultiStateMisfit
 method), 23
`grad()` (hippylib.modeling.misfit.MultPointwiseStateObservation
 method), 22
`grad()` (hippylib.modeling.misfit.PointwiseStateObservation
 method), 23
`grad()` (hippylib.modeling.prior._Prior method), 33

H

- `hessian()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` method), 49
- `hippylib` (module), 57
- `hippylib.algorithms` (module), 47
- `hippylib.algorithms.bfgs` (module), 37
- `hippylib.algorithms.cgsampler` (module), 38
- `hippylib.algorithms.cgsolverSteihaug` (module), 39
- `hippylib.algorithms.linalg` (module), 40
- `hippylib.algorithms.lowRankOperator` (module), 41
- `hippylib.algorithms.multivector` (module), 42
- `hippylib.algorithms.NewtonCG` (module), 35
- `hippylib.algorithms.randomizedEigensolve` (module), 43
- `hippylib.algorithms.randomizedSVD` (module), 45
- `hippylib.algorithms.steepestDescent` (module), 45
- `hippylib.algorithms.traceEstimator` (module), 46
- `hippylib.forward_uq` (module), 53
- `hippylib.forward_uq.parameter2QoiMap` (module), 47
- `hippylib.forward_uq.qoi` (module), 47
- `hippylib.forward_uq.taylorApproximationQoi` (module), 51
- `hippylib.forward_uq.varianceReductionMC` (module), 52
- `hippylib.mcmc` (module), 55
- `hippylib.mcmc.chain` (module), 53
- `hippylib.mcmc.diagnostics` (module), 53
- `hippylib.mcmc.kernels` (module), 53
- `hippylib.mcmc.tracers` (module), 54
- `hippylib.modeling` (module), 35
- `hippylib.modeling.expression` (module), 21
- `hippylib.modeling.misfit` (module), 21
- `hippylib.modeling.model` (module), 24
- `hippylib.modeling.modelVerify` (module), 27
- `hippylib.modeling.PDEProblem` (module), 19
- `hippylib.modeling.pointwiseObservation` (module), 27
- `hippylib.modeling.posterior` (module), 28
- `hippylib.modeling.prior` (module), 30
- `hippylib.modeling.reducedHessian` (module), 33
- `hippylib.modeling.timeDependentVector` (module), 35
- `hippylib.modeling.variables` (module), 35
- `hippylib.scheduling` (module), 55
- `hippylib.scheduling.collective` (module), 55
- `hippylib.utils` (module), 57
- `hippylib.utils.checkDolfinVersion` (module), 55
- `hippylib.utils.nb` (module), 56
- `hippylib.utils.parameterList` (module), 55
- `hippylib.utils.random` (module), 56
- `hippylib.utils.vector2function` (module), 56
- I**
- `init_parameter()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` method), 50
- `init_parameter()` (`hippylib.modeling.model.Model` method), 26
- `init_parameter()` (`hippylib.modeling.PDEProblem.PDEProblem` method), 20
- `init_parameter()` (`hippylib.modeling.PDEProblem.PDEVariationalProblem` method), 20
- `init_sample()` (`hippylib.mcmc.kernels.gpCNKernel` method), 54
- `init_sample()` (`hippylib.mcmc.kernels.ISKernel` method), 53
- `init_sample()` (`hippylib.mcmc.kernels.MALAKernel` method), 54
- `init_sample()` (`hippylib.mcmc.kernels.pCNKernel` method), 54
- `init_vector()` (`hippylib.algorithms.bfgs.RescaledIdentity` method), 38
- `init_vector()` (`hippylib.algorithms.linalg.DiagonalOperator` method), 40
- `init_vector()` (`hippylib.algorithms.linalg.Operator2Solver` method), 40
- `init_vector()` (`hippylib.algorithms.linalg.Solver2Operator` method), 40
- `init_vector()` (`hippylib.algorithms.lowRankOperator.LowRankOperator` method), 41
- `init_vector()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiHessian` method), 47
- `init_vector()` (`hippylib.modeling.posterior.GaussianLRPosterior` method), 28
- `init_vector()` (`hippylib.modeling.posterior.LowRankHessian` method), 29

`init_vector()` (hippylib.modeling.posterior.LowRankPosteriorSampler method), 29
`init_vector()` (hippylib.modeling.prior._BilaplacianR method), 32
`init_vector()` (hippylib.modeling.prior._BilaplacianRsolver method), 32
`init_vector()` (hippylib.modeling.prior._Prior method), 33
`init_vector()` (hippylib.modeling.prior._RinvM method), 33
`init_vector()` (hippylib.modeling.prior.GaussianRealPrior method), 30
`init_vector()` (hippylib.modeling.prior.LaplacianPrior method), 31
`init_vector()` (hippylib.modeling.prior.SqrtPrecisionPDE_Prior method), 32
`init_vector()` (hippylib.modeling.reducedHessian.FDHessian method), 33
`init_vector()` (hippylib.modeling.reducedHessian.ReducedHessian method), 34
`initialize()` (hippylib.modeling.timeDependentVector.TimeDependentVector method), 35
`inner()` (hippylib.algorithms.linalg.DiagonalOperator method), 40
`inner()` (hippylib.algorithms.linalg.Operator2Solver method), 40
`inner()` (hippylib.algorithms.linalg.Solver2Operator method), 40
`inner()` (hippylib.algorithms.lowRankOperator.LowRankOperator method), 41
`inner()` (hippylib.forward_uq.parameter2QoiMap.Parameter2QoiHessian method), 48
`inner()` (hippylib.modeling.posterior.LowRankHessian method), 29
`inner()` (hippylib.modeling.reducedHessian.FDHessian method), 34
`inner()` (hippylib.modeling.reducedHessian.ReducedHessian method), 34
`inner()` (hippylib.modeling.timeDependentVector.TimeDependentVector method), 35
`integratedAutocorrelationTime()` (in module hippylib.mcmc.diagnostics), 53
`ISKernel` (class in hippylib.mcmc.kernels), 53

K
`KLDistanceFromPrior()` (hippylib.modeling.posterior.GaussianLRPosterior method), 28

L
`LaplacianPrior` (class in hippylib.modeling.prior), 30
`LowRankHessian` (class in hippylib.modeling.posterior), 29
`LowRankOperator` (class in hippylib.algorithms.lowRankOperator), 41
`LowRankPosteriorSampler` (class in hippylib.modeling.posterior), 29
`LS_ParameterList()` (in module hippylib.algorithms.NewtonCG), 35

M
`MALAKernel` (class in hippylib.mcmc.kernels), 53
`MatAtB()` (in module hippylib.algorithms.linalg), 40
`MatMatMult()` (in module hippylib.algorithms.linalg), 40
`MatMvMult()` (in module hippylib.algorithms.multivector), 42
`MatMvTranspmult()` (in module hippylib.algorithms.multivector), 42
`MatPtAP()` (in module hippylib.algorithms.linalg), 40
`MCMC` (class in hippylib.mcmc.chain), 53
`Misfit` (class in hippylib.modeling.misfit), 22
`Model` (class in hippylib.modeling.model), 24
`modelVerify()` (in module hippylib.modeling.modelVerify), 27
`modelVerifyPlotErrors()` (in module hippylib.modeling.modelVerify), 27
`MollifiedBiLaplacianPrior()` (in module hippylib.modeling.prior), 31
`mpi_comm()` (hippylib.modeling.prior._BilaplacianR method), 32
`mult()` (hippylib.algorithms.linalg.DiagonalOperator method), 40
`mult()` (hippylib.algorithms.linalg.Solver2Operator method), 40
`mult()` (hippylib.algorithms.lowRankOperator.LowRankOperator method), 41
`mult()` (hippylib.forward_uq.parameter2QoiMap.Parameter2QoiHessian method), 48
`mult()` (hippylib.modeling.posterior.LowRankHessian method), 29
`mult()` (hippylib.modeling.prior._BilaplacianR method), 32
`mult()` (hippylib.modeling.prior._RinvM method), 33
`mult()` (hippylib.modeling.reducedHessian.FDHessian method), 34

`mult()` (`hippylib.modeling.reducedHessian.ReducedHessian` `method`), 34
`multi1_plot()` (in module `hippylib.utils.nb`), 57
`MultipleSerialPDEsCollective` (class in `hippylib.scheduling.collective`), 55
`MultiStateMisfit` (class in `hippylib.modeling.misfit`), 23
`MultiVector` (class in `hippylib.algorithms.multivector`), 42
`MultPointwiseStateObservation` (class in `hippylib.modeling.misfit`), 22
`MvDSmatMult()` (in module `hippylib.algorithms.multivector`), 43

N

`name()` (`hippylib.mcmc.kernels.gpCNKernel` `method`), 54
`name()` (`hippylib.mcmc.kernels.ISKernel` `method`), 53
`name()` (`hippylib.mcmc.kernels.MALAKernel` `method`), 54
`name()` (`hippylib.mcmc.kernels.pCNKernel` `method`), 54
`norm()` (`hippylib.modeling.timeDependentVector.TimeDependentVector` `method`), 35
`normal()` (`hippylib.utils.random.Random` `method`), 56
`normal_perturb()` (`hippylib.utils.random.Random` `method`), 56
`NullCollective` (class in `hippylib.scheduling.collective`), 55
`NullQoi` (class in `hippylib.mcmc.chain`), 53
`NullTracer` (class in `hippylib.mcmc.tracers`), 54

O

`Operator2Solver` (class in `hippylib.algorithms.linalg`), 40
`orthogonalize()` (`hippylib.algorithms.multivector.MultiVector` `method`), 43

P

`Parameter2QoiHessian` (class in `hippylib.forward_uq.parameter2QoiMap`), 47
`Parameter2QoiMap` (class in `hippylib.forward_uq.parameter2QoiMap`), 48
`parameter2QoiMapVerify()` (in module `hippylib.forward_uq.parameter2QoiMap`), 51
`parameter2QoiMapVerifyPlotErrors()` (in module `hippylib.forward_uq.parameter2QoiMap`), 51
`ParameterList` (class in `hippylib.utils.parameterList`), 55
`parRandom` (in module `hippylib.utils.random`), 56
`pCNKernel` (class in `hippylib.mcmc.kernels`), 54
`PDEProblem` (class in `hippylib.modeling.PDEProblem`), 19

`PDEVariationalProblem` (class in `hippylib.modeling.PDEProblem`), 20
`plot()` (in module `hippylib.utils.nb`), 57
`plot_eigenvalues()` (in module `hippylib.utils.nb`), 57
`plot_eigenvectors()` (in module `hippylib.utils.nb`), 57
`plot_pts()` (in module `hippylib.utils.nb`), 57
`plotEigenvalues()` (in module `hippylib.forward_uq.taylorApproximationQoi`), 52
`pointwise_variance()` (`hippylib.modeling.posterior.GaussianLRPosterior` `method`), 28
`pointwise_variance()` (`hippylib.modeling.prior._Prior` `method`), 33
`PointwiseStateObservation` (class in `hippylib.modeling.misfit`), 23
`proposal()` (`hippylib.mcmc.kernels.gpCNKernel` `method`), 54
`proposal()` (`hippylib.mcmc.kernels.ISKernel` `method`), 53
`proposal()` (`hippylib.mcmc.kernels.MALAKernel` `method`), 54
`proposal()` (`hippylib.mcmc.kernels.pCNKernel` `method`), 54

Q

`Qoi` (class in `hippylib.forward_uq.qoi`), 47
`QoiTracer` (class in `hippylib.mcmc.tracers`), 54
`qoiVerify()` (in module `hippylib.forward_uq.qoi`), 47
`qoiVerifyPlotErrors()` (in module `hippylib.forward_uq.qoi`), 47

R

`rademacher()` (`hippylib.utils.random.Random` `method`), 56
`rademacher_engine()` (in module `hippylib.algorithms.traceEstimator`), 46
`Random` (class in `hippylib.utils.random`), 56
`rank()` (`hippylib.scheduling.collective.MultipleSerialPDEsCollective` `method`), 55
`rank()` (`hippylib.scheduling.collective.NullCollective` `method`), 55
`reason` (`hippylib.algorithms.cgsolverSteihaug.CG Solver Steihaug` `attribute`), 39
`reduced_eval()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 50
`reduced_gradient()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `method`), 50
`ReducedHessian` (class in `hippylib.modeling.reducedHessian`), 34

ReducedSpaceNewtonCG (class in hip-
 pylib.algorithms.NewtonCG), 35
 ReducedSpaceNewtonCG_ParameterList() (in
 module hippylib.algorithms.NewtonCG), 37
 RescaledIdentity (class in hip-
 pylib.algorithms.bfgs), 38
 retrieve() (hippylib.modeling.timeDependentVector.TimeDependentVector
 method), 35
 Rsolver() (hippylib.modeling.model.Model method),
 24
 run() (hippylib.mcmc.chain.MCMC method), 53
S
 sample() (hippylib.algorithms.cgsampler.CGSampler
 method), 39
 sample() (hippylib.mcmc.kernels.gpCNKernel
 method), 54
 sample() (hippylib.mcmc.kernels.ISKernel method), 53
 sample() (hippylib.mcmc.kernels.MALAKernel
 method), 54
 sample() (hippylib.mcmc.kernels.pCNKernel method),
 54
 sample() (hippylib.modeling.posterior.GaussianLRPosterior
 method), 29
 sample() (hippylib.modeling.posterior.LowRankPosteriorSampler
 method), 29
 sample() (hippylib.modeling.prior._Prior method), 33
 sample() (hippylib.modeling.prior.GaussianRealPrior
 method), 30
 sample() (hippylib.modeling.prior.LaplacianPrior
 method), 31
 sample() (hippylib.modeling.prior.SqrtPrecisionPDE_Prior
 method), 32
 SampleStruct (class in hippylib.mcmc.chain), 53
 set_H0inv() (hippylib.algorithms.bfgs.BFGS_operator
 method), 38
 set_operator() (hip-
 pylib.algorithms.cgsampler.CGSampler
 method), 39
 set_operator() (hip-
 pylib.algorithms.cgsolverSteihaug.CGSolverSteihaug
 method), 40
 set_preconditioner() (hip-
 pylib.algorithms.cgsolverSteihaug.CGSolverSteihaug
 method), 40
 set_TR() (hippylib.algorithms.cgsolverSteihaug.CGSolverSteihaug
 method), 39
 setLinearizationPoint() (hip-
 pylib.forward_uq.parameter2QoiMap.Parameter2QoiMap
 method), 50
 setLinearizationPoint() (hip-
 pylib.forward_uq.qoi.Qoi method), 47
 setLinearizationPoint() (hip-
 pylib.modeling.misfit.ContinuousStateObservation
 method), 22
 setLinearizationPoint() (hip-
 pylib.modeling.misfit.Misfit method), 22
 setLinearizationPoint() (hip-
 pylib.modeling.misfit.MultiStateMisfit method),
 23
 setLinearizationPoint() (hip-
 pylib.modeling.misfit.MultiPointwiseStateObservation
 method), 23
 setLinearizationPoint() (hip-
 pylib.modeling.misfit.PointwiseStateObservation
 method), 23
 setLinearizationPoint() (hip-
 pylib.modeling.PDEProblem.PDEProblem
 method), 20
 setLinearizationPoint() (hip-
 pylib.modeling.PDEProblem.PDEVariationalProblem
 method), 20
 setPointForHessianEvaluations() (hip-
 pylib.modeling.model.Model method), 26
 SetToOwnedGid() (in module hip-
 pylib.algorithms.linalg), 40
 show_solution() (in module hippylib.utils.nb), 57
 showMe() (hippylib.utils.parameterList.ParameterList
 method), 56
 singlePass() (in module hip-
 pylib.algorithms.randomizedEigensolver),
 44
 singlePassG() (in module hip-
 pylib.algorithms.randomizedEigensolver),
 44
 singlePassSVD() (in module hip-
 pylib.algorithms.randomizedSVD), 45
 size() (hippylib.scheduling.collective.MultipleSerialPDEsCollective
 method), 55
 size() (hippylib.scheduling.collective.NullCollective
 method), 55
 solve() (hippylib.algorithms.bfgs.BFGS method), 37
 solve() (hippylib.algorithms.bfgs.BFGS_operator
 method), 38
 solve() (hippylib.algorithms.bfgs.RescaledIdentity
 method), 38
 solve() (hippylib.algorithms.cgsolverSteihaug.CGSolverSteihaug
 method), 40
 solve() (hippylib.algorithms.linalg.Operator2Solver
 method), 40
 solve() (hippylib.algorithms.lowRankOperator.LowRankOperator
 method), 41
 solve() (hippylib.algorithms.NewtonCG.ReducedSpaceNewtonCG
 method), 37
 solve() (hippylib.algorithms.steepestDescent.SteepestDescent
 method), 46
 solve() (hippylib.modeling.posterior.LowRankHessian
 method), 29

`solve()` (`hippylib.modeling.prior._BilaplacianRsolver` `termination_reasons` (hippylib.algorithm.solver.BilaplacianRsolver), 32
`solveAdj()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `pylib.algorithm.solver.NewtonCG.ReducedSpaceNewtonCG` (hippylib.algorithm.solver.NewtonCG.ReducedSpaceNewtonCG), 37
`solveAdj()` (`hippylib.modeling.model.Model` `termination_reasons` (hippylib.algorithm.solver.steepestDescent.SteepestDescent), 26
`solveAdj()` (`hippylib.modeling.PDEProblem.PDEProblem` `TimeDependentVector` (class in hippylib.modeling.timeDependentVector), 20
`solveAdj()` (`hippylib.modeling.PDEProblem.PDEVariationalProblem` `TimeDependentVector` (in module hippylib.algorithm.solver.linalg), 21
`solveAdjIncremental()` (`hippylib.algorithm.solver.NewtonCG` `TR_ParameterList()` (in module hippylib.algorithm.solver.NewtonCG), 37
`solveAdjIncremental()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `LowRankOperator` (hippylib.algorithm.solver.LowRankOperator), 50
`solveAdjIncremental()` (`hippylib.modeling.model.Model` `trace()` (hippylib.modeling.posterior.GaussianLRPosterior), 27
`solveFwd()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `trace()` (in module hippylib.algorithm.solver.linalg), 50
`solveFwd()` (`hippylib.modeling.model.Model` `trace2()` (hippylib.algorithm.solver.LowRankOperator), 27
`solveFwd()` (`hippylib.modeling.PDEProblem.PDEProblem` `trace_update()` (hippylib.modeling.posterior.GaussianLRPosterior), 20
`solveFwd()` (`hippylib.modeling.PDEProblem.PDEVariationalProblem` `TraceEstimator` (class in hippylib.algorithm.solver.traceEstimator), 21
`solveFwdIncremental()` (`hippylib.forward_uq.parameter2QoiMap.Parameter2QoiMap` `TraceEstimator` (in module hippylib.algorithm.solver.linalg), 51
`solveFwdIncremental()` (`hippylib.modeling.model.Model` `TrueHessian()` (hippylib.modeling.reducedHessian.ReducedHessian), 27
`solveIncremental()` (`hippylib.modeling.PDEProblem.PDEProblem` `TraceEstimator` (hippylib.algorithm.solver.traceEstimator), 20
`solveIncremental()` (`hippylib.modeling.PDEProblem.PDEVariationalProblem` `uniform()` (hippylib.utils.random.Random), 21
`Solver2Operator` (class in hippylib.algorithm.solver.linalg), 40
`speckle()` (`hippylib.utils.random.Random` `update()` (hippylib.algorithm.solver.bfgs.BFGS_operator), 56
`SqrtPrecisionPDE_Prior` (class in hippylib.modeling.prior), 31
`SteepestDescent` (class in hippylib.algorithm.solver.steepestDescent), 45
`SteepestDescent_ParameterList()` (in module hippylib.algorithm.solver.steepestDescent), 46
`store()` (`hippylib.modeling.timeDependentVector.TimeDependentVector` `update_x_with_TR()` (hippylib.algorithm.solver.cgsolverSteihaug.CGSolverSteihaug), 35

T
`TaylorApproximationQoi` (class in hippylib.forward_uq.taylorApproximationQoi), 51
`termination_reasons` (hippylib.algorithm.solver.bfgs.BFGS attribute), 38

U
`uniform()` (hippylib.utils.random.Random), 56
`update()` (hippylib.algorithm.solver.bfgs.BFGS_operator), 38
`update_x_with_TR()` (hippylib.algorithm.solver.cgsolverSteihaug.CGSolverSteihaug), 40
`update_x_without_TR()` (hippylib.algorithm.solver.cgsolverSteihaug.CGSolverSteihaug), 40

V
`variance()` (hippylib.forward_uq.taylorApproximationQoi.TaylorApproximationQoi), 52
`varianceReductionMC()` (in module hippylib.forward_uq.varianceReductionMC), 52
`vector2Function()` (in module hippylib.utils.vector2function), 56

W
`write_vtk()` (in module hippylib.modeling.pointwiseObservation), 28

Z

`zero()` (*hippylib.modeling.timeDependentVector.TimeDependentVector*
method), [35](#)