hIPPYfire:(yet-to-be-finalized)

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Omar Ghattas, Supervisor

Dr. Omar Ghattas

Dr. Umberto Villa

hIPPYfire:(yet-to-be-finalized)

by

Karan Prakash Hiranandani

REPORT

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

The University of Texas at Austin

May 2023

A short dedication to someone special.

Acknowledgments

Acknowledgments are technically optional, but come on, it takes a village. Say thanks! This section is not limited to a single page.

This LaTeX template was originally written in 1991 by Young U. Ryu and later modified by Miguel A. Lerma and Craig McCluskey. Thanks you guys, we all owe you. The template was heavily modified by Shane A. McQuarrie in 2023.

hIPPYfire:(yet-to-be-finalized)

by

Karan Prakash Hiranandani, M.S.
The University of Texas at Austin, 2023
Supervisor: Dr. Omar Ghattas

This study presents the implementation of a large-scale Bayesian and deterministic inverse problem solver named hippyfire.

Table of Contents

List of Figures

Chapter 1

Introduction

The developments in mathematical computing have facilitated the solutions of analyses of *forward problems*. A forward pr The readily available computational resources and the conception and implementation of efficient numerical algorithms

Similar to The FEniCS Project's DOLFIN library [1, 13], Firedrake [15] is a library that provides automated solutions of partial differential equations (PDEs) using the finite element method (FEM). The philosophy behind the development of these frameworks was separation of concerns. Since a multidisciplinary skillset, which ranges from mathematical expertise in numerical analyses to a deep understanding of parallel computation, is required for the development of these tools, it became more practical to develop abstract layers in the library that catered to a particular skillset.

Chapter 2

Theory Behind Inverse Problems

This section discusses two popular techniques adopted to solve inverse problems—namely the determinstic framework [12, 26] and Bayesian framework [7, 19] for inverse problems. `hIPPYfire` currently implements an infinite-dimensional Bayesian framework [6], and consequently, additional emphasis has been placed on the theory of the latter framework.

Lower-case italic font is used to represent scalar valued functions like the parameter m, state u, observed data d etc. The discretized equivalents of these functions $\mathbb{R}^n$ (where n is the discretization dimension) are denoted with a bold, lowercase font, such as $\mathbf{m}$, $\mathbf{u}$, $\mathbf{d}$ etc. Vector functions in $\mathbb{R}^2$ or $\mathbb{R}^3$ are denoted in italic, bold, lowercase font, such as $\boldsymbol{v}$ (velocity field). Infinite-dimensional spaces are represented through calligraphic font, such as $\mathcal{A}$. Scalars are denoted by using Greek font.

2.1 Deterministic Inversion

The solution of an inverse problem involves the inference of the parameter $m \in \mathcal{X}$, given data $\boldsymbol{d} \in \mathbb{R}^q$ by using the following *parameter-to-observable* map:

$$\mathcal{F}(m) = d + \eta \tag{2.1.1}$$

Similar to *hIPPYlib*, the linear or non-linear *parameter-to-observable (p2o)* map is depicted as $\mathcal{F} : \mathcal{M} \to \mathbb{R}^q$. Note that $\mathcal{M} \subseteq L^2(\mathcal{D})$, where $\mathcal{D} \subset \mathbb{R}^d$ is a bounded domain. It is important to distinguish between the *p2o* map the forward problem. The forward problem is a map from the parameter m to the PDEs that govern the physical system, i.e., $m \to r(u, m)$, where $r(u, m)$ is the residual and $u \in \mathcal{V}$ is a state variable and $\mathcal{V}$ is a Hilbert Space of functions defined on $\mathcal{V}$.

The forward problem is one of the three components of the forward map $\mathcal{F}$. The second component is the map $r : \mathcal{V} \times \mathcal{M} \to \mathcal{V}^*$. This map involves the solution of the governing PDEs. The third component is the observation operator (B), which maps u to the observable

$y \in \mathbb{R}^q$, which can be high or infinite dimensional. Thus, the map $\mathcal{F}$ is now defined as:

$$\mathcal{F}(m) = \mathcal{B}(u), given, r(u, m) = 0 \tag{2.1.2}$$

The noise, η, accounts for the difference between the observable y and data $\boldsymbol{d}$ and is a modelled as a Gaussian centred at 0 with a covariance Γ_{noise}. The source of the noise can be traced to imprecise measurements, model, and/or numerical errors. Although their exact values are not known, their statistical information (mean, variance, etc.), are known.

The ill-posedness of inverse problems [21], as discussed earlier, can primarily be attributed to their instability. The collapse of the spectrum of $\mathcal{F}$ is the main culprit behind this phenomenon. The small eigenvalues of $\mathcal{F}$, which correspond to eigen function modes below the noise threshold, cause the noise in the data to blow up exponentially—rendering the inversion useless. The most obvious solution to this problem would be to discard the irrelevant eigenvalues. Although a truncated SVD presents a viable solution to the problem, the *Tikhonov Regularization* technique [11] allows us to formulate the inverse problem as an optimization problem instead of applying a filter. This can be mathematically represented as a non-linear least squares optimization problem:

$$\min_{m \in \mathcal{M}} \mathcal{J}(m) := \frac{1}{2} ||\mathcal{F}(m) - \boldsymbol{d}||^2_{\Gamma^{-1}_{noise}} + \mathcal{R}(m) \tag{2.1.3}$$

$\mathcal{J}(m)$ represents the cost functional. The first term on the RHS represents the misfit between $\mathcal{F}(m)$, weighted by the inverse noise covariance η^{-1}_{noise}, and data $\boldsymbol{d}$. The regularization parameter, $\mathcal{R}(m)$, ensures smoothness on the inversion parameter m.

Information on the first and second derivatives (gradient and Hessian, respectively) is required to solve the nonlinear optimization problem. Lagrangian techniques [22] were adopted in the development of the `hIPPYlib` library [23] to compute the actions of the Hessian and gradient. `hIPPYfire` makes use of the *Inexact Newton Conjugate Gradient Method* to solve the optimization problem, similar to the flow followed by `hIPPYlib` [23].

The regularization technique of solving inverse problems (2.1.1), although scalable with efficient inverse solvers, fails to account for uncertainties in the inferred parameters. This can be attributed to the fact that this approach only provides a point estimate of the inverse problem. Thus, it is not very useful for ill-posed problems with non-negligible noise—thereby prompting the conception of the Bayesian framework.

2.2 Bayesian Inversion

The Bayesian framework treats the inverse problem as statistical inference over a space of uncertain parameters. It computes the probability of the parameter being conditioned on the data as a *posterior probability distribution.* This framework accounts for any assumptions and constraints on the parameter before data collection and depicts them through a *prior distribution.* This is combined with the *likelihood* that the observed data could be obtained from a given set of parameters. This is achieved by analyzing the posterior through mean estimation, sample drawing, analysis of the covariance etc.

However, complete characterization of the posterior is impractical for expensive PDE forward models, especially for ones in high dimensions that have been obtained after the discretization of infinite-dimensional parameter fields. Recently developed techniques, however, take advantage of the low dimensionality to address these problems—similar to the properties of the deterministic framework. Some of these include forward model reduction [10], Markov chain Monte-Carlo techniques that utilize the log-likelihood Hessian approximations [14], randomize-then-optimize techniques [27], etc. `hIPPYfire`, much like its predecessor [23], uses *Lagrangian approximation* of the posterior. This is extremely scalable and efficient—especially if the same properties that made the regularized Newton-CG method so powerful are used.

The infinite-dimensional Baye's formula is given as:

$$\frac{d\mu_{post}}{d\mu_{prior}} \propto \pi_{like}(\mathbf{d}|m) \tag{2.2.1}$$

The LHS and RHS represent the Radon-Nikodym derivative [28] and likelihood, respectively. To study the impact of the noise on the likelihood, the noise is modelled as a centered Gaussian on $\mathbb{R}^q$ with a covariance of Γ_{noise}. Thus, the likelihood can be expressed as

$$\pi_{like}(\mathbf{d}|m) \propto exp(-\Phi(m)) \tag{2.2.2}$$

where $\Gamma(m) = \frac{1}{2}||\mathcal{F}(m) - \boldsymbol{d}||^2_{\Gamma^{-1}_{noise}}$ denotes the negative log-likelihood. The prior is chosen to be Gaussian, giving the following relation:

$$d\mu_{prior}(m) \propto exp\{-\frac{1}{2}||m - m_{pr}||^2_{C^{-1}_{prior}}\} \tag{2.2.3}$$

where $m \sim \mathcal{N}(m_{pr}, C_{prior})$. The prior covariance C_{prior} smooths out the parameter (similar to the action performed by the regularization term in 2.1) to ensure that the rough components of the data are computed prior to the computation of the result of a well-posed Bayesian inversion. The construction of the prior is similar to the methodology followed for the same by `hIPPYlib` [23]. The equations of the likelihood (Eqn. (2.2.2)) and prior (Eqn.(2.2.3)) are substituted in the posterior distribution equation (Eqn. (2.2.1)) to yield:

$$d\mu_{post} \propto exp\{-\frac{1}{2}||\mathcal{F}(m) - \boldsymbol{d}||^2_{\Gamma^{-1}_{noise}} - \frac{1}{2}||m - m_{pr}||^2_{C^{-1}_{prior}}\} \tag{2.2.4}$$

The next step in the flow involves the computation of the MAP point m_{MAP}—which is the parameter field that maximizes the posterior distribution. It is obtained by solving the following optimization problem:

$$m_{MAP} := \underset{m \in \mathcal{M}}{argmin} \frac{1}{2}||\mathcal{F}(m) - \boldsymbol{d}||^2_{\Gamma^{-1}_{noise}} + \frac{1}{2}||m - m_{pr}||^2_{C^{-1}_{prior}} \tag{2.2.5}$$

It is noteworthy that the prior performs the role of *Tikhonov Regularization* [11]. Similaritires can be drawn between Eqn. (2.2.5) and Eqn. (2.1.3). In case the *p2o* map is non-linear, the posterior does not follow a Gaussian distribution. However, certain assumptions can be made on the noise covariance Γ_{noise}, number q of observations, and regularity of the *p2o* map $\mathcal{F}$, the Laplace approximation can be utilized to estimate the expected value of the prior [20, 8]. This is followed by the discretization of the Bayesian inverse problem, which has been explained in [6, 23]. The complete implementation of the Laplace approximation, along with the Bayesian discretization, in `hIPPYfire` is similar to that of `hIPPYlib` [23].

Chapter 3

Software Framework

3.1 Firedrake

As mentioned previously, one of the improvements Firedrake [15] implemented over FEniCS [1] is the creation of a new abstraction layer, namely PyOP2 [16], to distinguish between the local discretization or mathematical operators and their parallel execution over the mesh in the implementation layer. Firedrake models a finite problem as a combination of several abstractions—the number of which is greater than previously known models. The universal

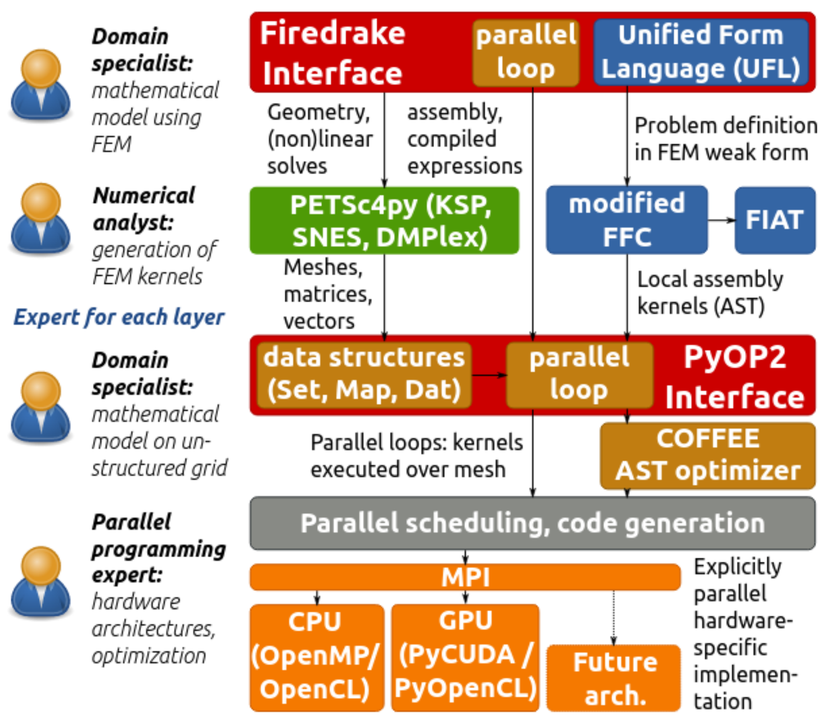

Figure 3.1: Firedrake Abstractions
Depiction of the separation of user concerns in Firedrake.Tools using FEniCS and PETSc are highlighted in blue and green respectively. The PyOP2 layers are shown in brown, while the backend engine is shown in orange [15].

nature of UFL [2] allows Firedrake to use it seamlessly. The cost of typical finite element problems can be attributed to data movement and floating point operations—both of which are proportional to the mesh size. These operations can be divided into two categories—custom mesh-defined data structure iterations and sparse linear algebra. While Firedrake utilizes PETSc [3] for the latter, PyOP2 was designed to address the former [16]. Additional information regarding the integration of PyOP2 with the Firedrake layer is shown in the figure below [15].

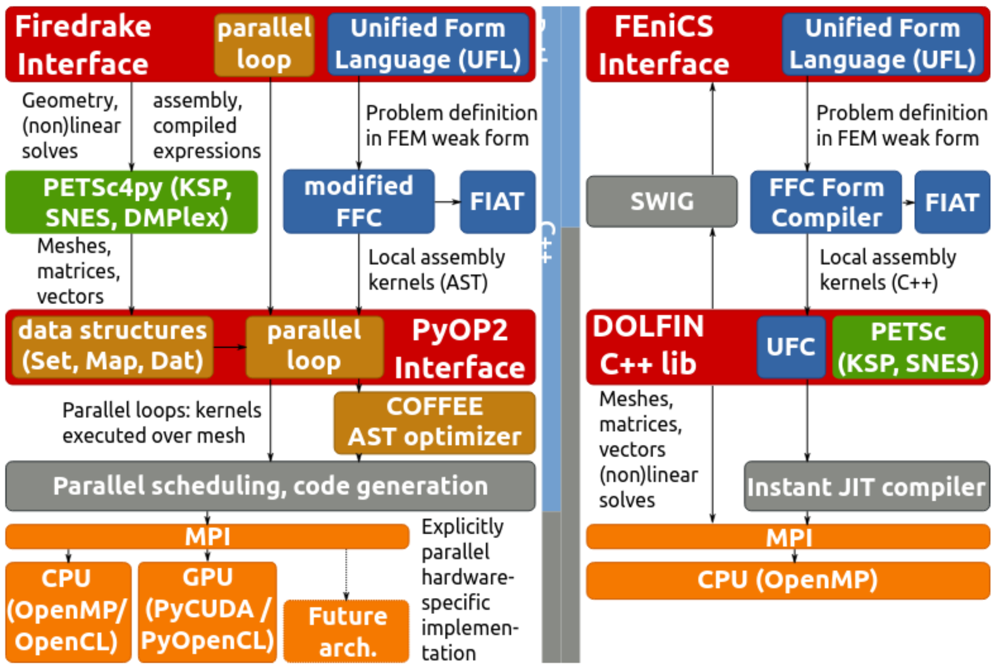

Figure 3.2: Firedrake v/s FEniCS flow
The clear distinction introduced in Firedrake is evident from the PyOP2 interface in its flow. [17].

The primary motivation behind using Firedrake can be attributed to its improved abstraction layer. However, a comparative analysis of the performances of Firedrake and FEniCS was conducted [15]. Firedrake reported a better performance than FEniCS—however, no concrete reason has been provided for its superior performance.

3.2 `hIPPYfire`

`hIPPYfire` attempts to accomplish the same objective as that of `hIPPYlib` , i.e., implementation of scalable algorithms for PDE-based deterministic and Bayesian inverse problems. However, unlike its predecessor, it is built on Firedrake instead of FEniCS. The user is required to provide the PDE problem and likelihood in UFL [2], and `hIPPYfire` computes the gradient and Hessian. `hIPPYfire` is currently in development and does not support all the functionality of `hIPPYlib` at the timing of writing this report. Its different components have been summarized below:

- **Models**: The `modeling` module allows the user to specify information on the forward problem, misfit functional, and the prior.

   1. Forward Problem: This module attempts to solve forward, adjoint, and incremental problems. `hIPPYlib` and `hIPPYfire` both accept user input for the forward problem as a UFL form or user-defined object. The latter is required for transient inverse problems. However, if the forward problem is input as a UFL form, `hIPPYfire` computes the gradient and Hessian as well.
   2. Misfit: The misfit module evaluates the negative log-likelihood and its derivatives. Currently, the only misfit functional `hIPPYfire` provides support for is that of continuous observations—however, other functionals are currently in development.
   3. Prior: The prior computes the negative log-density and its derivatives, in addition to drawing samples and estimating the marginal variance. The user can select a Bilaplacian prior in `hIPPYfire`'s current implementation. There is a provision to accept user-defined priors as well.
   4. Model: The model is used to set up the *p2o* map. Its three components are computed from the abovementioned modules.
   5. Hessian: If the forward problem is input in a standard UFL form, `hIPPYfire` internally computes the Hessian of the forward map. The collapse of the spectrum of the Hessian significantly influences the ill-posedness of the problem. However, the Hessian assumes a dense structure after discretization, thereby requiring forward and adjoint solves. Since the dimension of the Hessian is equal to that of the parameter, computing the Hessian for large-scale problems is not feasible. The rapidly decaying spectrum of the Hessian is exploited because the eigenvalues that tend to zero contain minimal information about the infinite-dimensional parameter field [9, 5].

  In case of transient problems, the user will have to provide their own derivatives.

- **Algorithms**: The `algorithms` module contains an implementation of the inexact Newton-CG algorithm [4]. This is used to solve the deterministic inversion problem and compute the mean of the approximate posterior distribution (also called the MAP point) for Bayesian inversion problems. Following the computation of the gradient and Hessian forms, the Newton system is solved according to the Steihaug criterion [18]. Additional information regarding the implementation of this algorithm can be found in Villa et al. [25]. Custom linear algebra algorithms have also been implemented to perform basic matrix-vector operations that have been described later in this section.

- **Utilities**: The `utils` module contains certain helper functions to extract relevant data. The `vector2function` module wraps a discrete vector onto a continuous function, while the `rand` module generates random functions that are used to model the noise function and vector. The `parameterlist` module creates a custom list of parameters for the Newton solver.

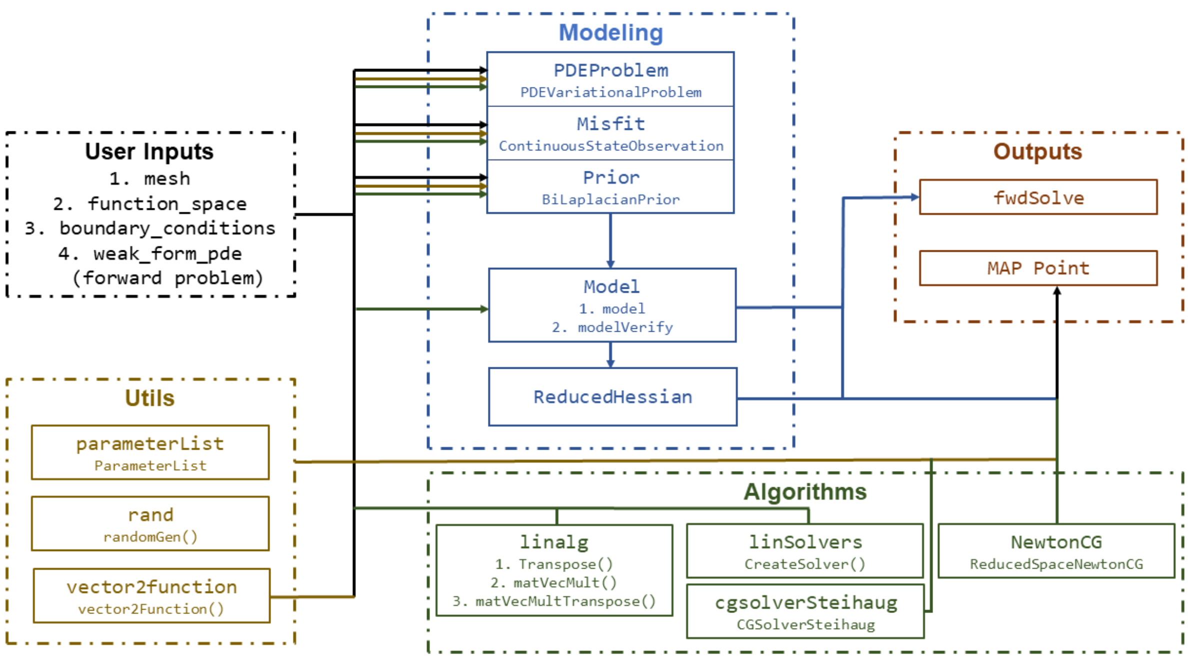

Figure 3.3: Flow and functionality of `hIPPYfire`

One of the major challenges faced in the development of `hIPPYfire` was the limited functionality of Firedrake's `firedrake.matrix.Matrix` data structure. Certain applications required matrix transpose and matrix-vector multiplication operations. Since these operations are not defined for firedrake Matrix, Firedrake's interface with PETSc was used to create linear algebra operations between Firedrake objects and their PETSc wrappers. These operations include matrix transpose (`Transpose()`, matrix-vector multiplication (`matVecMult()`), and matrix-transpose-vector multiplication (`matVecMultTranspose()`)—all of which have been defined in the `linalg` module. While `hIPPYfire` provides APIs to create different kinds of solvers, `hIPPYfire`'s `linSolvers` module allows the creation of custom solvers that can emulate `hIPPYlib`'s different solvers by passing different parameters.

Chapter 4

Sample Problem

In order to validate the different modules of `hIPPYfire`, one of `hIPPYlib`'s Bayesian inversion test cases, namely `3-SubsurfaceBayesian` [24], was recreated. This test case utilized all of `hIPPYfire`'s models that have been currently developed. Additional test cases shall be added for future functionality. The test case is briefly discussed below; a detailed description of the test case can be found in `hIPPYlib`'s repository [24].

This test case attempts to quantify the uncertainty in a Bayesian inversion problem, as described in Section 2. The forward problem of this test case is governed by an elliptic PDE. The objective is to compute the parameter fields with a certain probability that these gave rise to the observed data. Please note that the following test case admits discretized expressions of the parameter space, i.e., they are finite-dimensional.

A Gaussian prior is assumed with a mean of $\mathbf{m}_{prior}$. Its covariance, Γ_{prior}, is obtained by discretizing the inverse of the differential operator $\mathcal{A}^{-2} = (-\gamma\Delta + \delta\mathbf{I})^{-2}$, where $\gamma, \delta > 0$. The prior is chosen such that the well-posedness of this problem is ensured.

The likelihood is calculated as shown below:

$$\mathbf{d}_{obs} = \mathbf{f(m)} + \mathbf{e} \tag{4.0.1}$$

$\mathbf{e}$ represents the noise, which is computed by the `randomGen()` method in the `rand` module.

$$\pi_{like}(\mathbf{d}_{obs}|\mathbf{m}) = exp(-\frac{1}{2}(\mathbf{f(m)} - \mathbf{d}_{obs})^T \Gamma_{noise}^{-1}(\mathbf{f(m)} - \mathbf{d}_{obs})) \tag{4.0.2}$$

The above equation presents a discretized version of the likelihood function. The mean of the posterior distribution, also referred to as m_{MAP}, is the parameter vector that maximizes the posterior. Its discretized computation is listed below:

$$\mathbf{m}_{MAP} := \underset{m}{argmin} \mathcal{J}(\mathbf{m}) := (\frac{1}{2}||\mathbf{f(m)} - d_{obs}||^2_{\Gamma^{-1}_{noise}} + \frac{1}{2}||\mathbf{m} - \mathbf{m}_{prior}||^2_{\Gamma^{-1}_{noise}} \tag{4.0.3}$$

Problem flow has been defined until the computation of the MAP point in `hIPPYfire`. A few code snippets have been included below

- **Mesh and FEM setup**: A two-dimensional unit square mesh is created with a P2 finite element space for `state` and `adjoint` variables and P1 for `parameter`.

```
        ndim = 2
        nx = 64
        ny = 64
        mesh = fd.UnitSquareMesh(nx, ny)
        Vh2 = fd.FunctionSpace(mesh, 'Lagrange', 2)
        Vh1 = fd.FunctionSpace(mesh, 'Lagrange', 1)
        Vh = [Vh2, Vh1, Vh2]

```

- **Generating the true parameter**: Generate a random vector field for the true parameter

```
        mtrue = randomGen(Vh[STATE])

```

- **Forward Problem**: As mentioned in Section 3, the `PDEVariationalProblem` class sets up the forward problem component of the *p2o* map. In addition to the finite element components defined above, it requires an expression of the weak form of the PDE (given by `pde_varf`) and boundary conditions for the forward (`bc`) and incremental and adjoint problems (`bc0`). The `PDEVariationalProblem` class solves the forward/adjoint and incremental problems and computes the relevant partial derivatives with respect to the state, parameter, and adjoint variables.

```
        u_bdr = fd.SpatialCoordinate(mesh)[1]
        u_bdr0 = fd.Constant(0.0)

        bc = fd.DirichletBC(Vh[STATE], u_bdr, [3, 4]) # [3, 4] indicates that bc is applied to y == 0 and y ==1
        bc0 = fd.DirichletBC(Vh[STATE], u_bdr0, [3, 4])

        f = fd.Constant(1.0)

        def pde_varf(u, m, p):
            return ufl.exp(m) * ufl.inner(ufl.grad(u), ufl.grad(p)) * ufl.dx - f * p * ufl.dx

        pde = PDEVariationalProblem(Vh, pde_varf, bc, bc0, is_fwd_linear=True)

```

  The `is_fwd_linear=True` flag allows the user to set a non-linear forward map as well.

- **Prior setup**: The class `BiLaplacianPrior` creates a Gaussian prior with zero average, Additional information regarding the covariance can be found in Villa et al. [24].

```
        pr = BiLaplacianPrior(Vh[PARAMETER], gamma, delta, robin_bc=True)
        x = fd.SpatialCoordinate(mesh)
        mtrue = fd.interpolate(fd.sin(x[0])*fd.cos(x[1]), Vh[PARAMETER]).vector()
        m0 = fd.interpolate(fd.sin(x[0]), Vh[PARAMETER]).vector()
        objs = [fd.Function(Vh[PARAMETER], mtrue), fd.Function(Vh[PARAMETER], pr.mean)]

```

The variables `gamma, delta` are input by the user. Plots of the vectors `mtrue` and `pr.mean` are generated and shown below. For the purpose of validation through this test case, `mtrue` is a known function and not randomly generated, as in `hIPPYlib` [24]

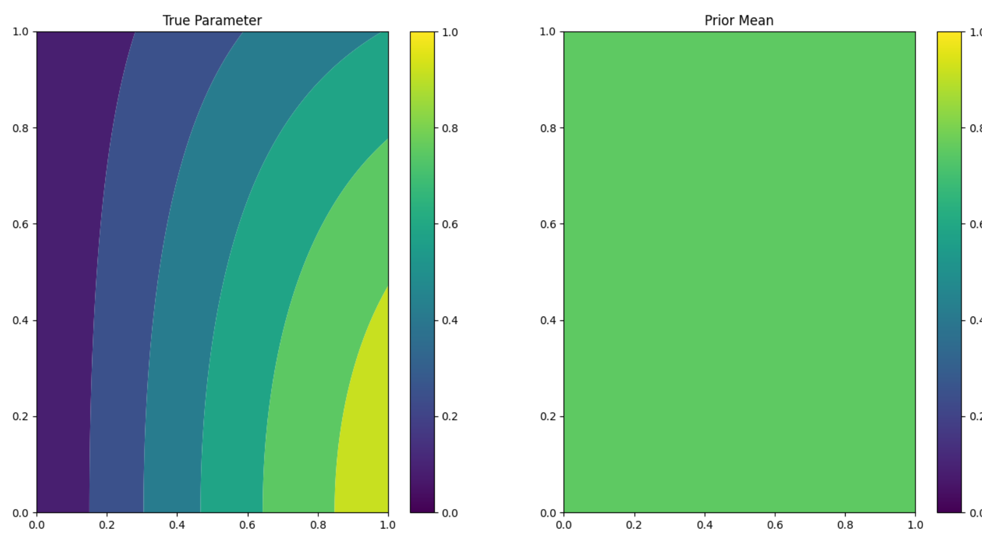

Figure 4.1: Variation of the true parameter and mean.

- **Misfit**: `hIPPYfire` currently provides support for `ContinuousStateObservation`, which sets up the observation parameter $\mathcal{B}$. The observables which shall provide our input data are first generated by solving the forward problem by using the true parameter $\mathbf{m}_{true}$

```
        misfit = ContinuousStateObservation(Vh[STATE], ufl.dx, bcs=bc0)
        misfit.noise_variance = 1e-4
        utrue = pde.generate_state()
```

```
        x = [utrue, mtrue.vector(), None]
        pde.solveFwd(x[STATE], x)
        misfit.d.axpy(1., utrue)
        misfit.d.axpy(float(np.sqrt(misfit.noise_variance)),
    randomGen(Vh[STATE]).vector())

```

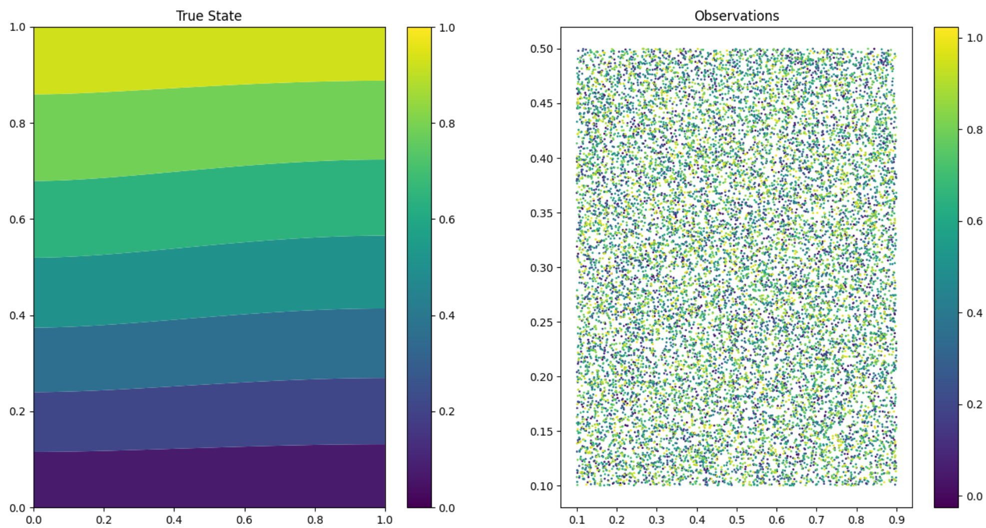

Figure 4.2: Variation of the true state and the observations.

- **Model**: The model, which is created by the `model` class, depends on three components—namely `PDEVariationalProblem`, `misfit`, and `prior`. The `PDEVariationalProblem` provides solutions for the forward and adjoint problems and incremental forward and adjoint problems. The `prior` applies the regularization operator to a vector, while the `misfit` computes the cost functional and partial derivatives with respect to the state and parameter variables. Forward finite differences are used to test the model through the `modelVerify` module.

```
        model = Model(pde, pr, misfit)
        eps, err_grad, err_H = modelVerify(model, m0, misfit_only=
    False)

```

```
        (yy, H xx) - (xx, H yy) =  0.0

```

- **MAP Point**: The Newton-CG method is used to compute the MAP point.

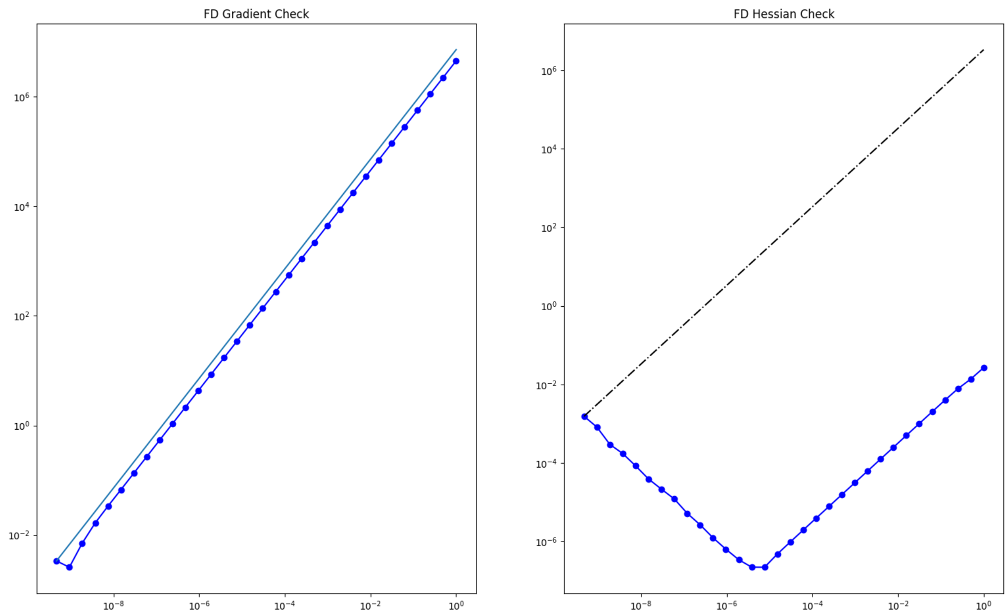

Figure 4.3: Gradient and Hessian Checks obtained from `modelVerify`

```
        m = pr.mean.copy()
        solver = ReducedSpaceNewtonCG(model)
        solver.parameters["rel_tolerance"] = 1e-6
        solver.parameters["abs_tolerance"] = 1e-12
        solver.parameters["max_iter"]      = 25
        solver.parameters["GN_iter"] = 5
        solver.parameters["globalization"] = "LS"
        solver.parameters["LS"]["c_armijo"] = 1e-4
        x = solver.solve([None, m, None])

```

The following output was obtained:

```
        Relative/Absolute residual less than tol
Converged in  19  iterations with final norm  5.65259722013104e-08

It  cg_it cost            misfit          reg              (g,dm)
        ||g||L2        alpha          tolcg
1   1    9.452277e-01    8.990612e-01    4.616652e-02    -2.456844e+01
      1.685316e+02   1.000000e+00   5.000000e-01
2   2    4.228824e-01    3.488744e-01    7.400803e-02    -1.043988e+00
      1.715492e+01   1.000000e+00   3.190463e-01
3   5    4.027649e-01    3.196818e-01    8.308311e-02    -4.042007e-02
      3.036538e+00   1.000000e+00   1.342297e-01
4   7    4.026241e-01    3.196381e-01    8.298594e-02    -3.009693e-04
      2.404168e-01   1.000000e+00   3.776954e-02
```

```
5   7    4.026228e-01    3.196101e-01    8.301271e-02    -2.859017e-06    1.569884e-02   1.000000e+00   9.651461e-03
6   8    4.026228e-01    3.196134e-01    8.300935e-02    -4.118272e-08    8.007385e-04   1.000000e+00   2.179740e-03
5.941225051879883 Executiion time

Converged in  6  iterations.
Termination reason:  Norm of the gradient less than tolerance
Final gradient norm:  7.784347671669573e-06
Final cost:  0.40262278305284716

```

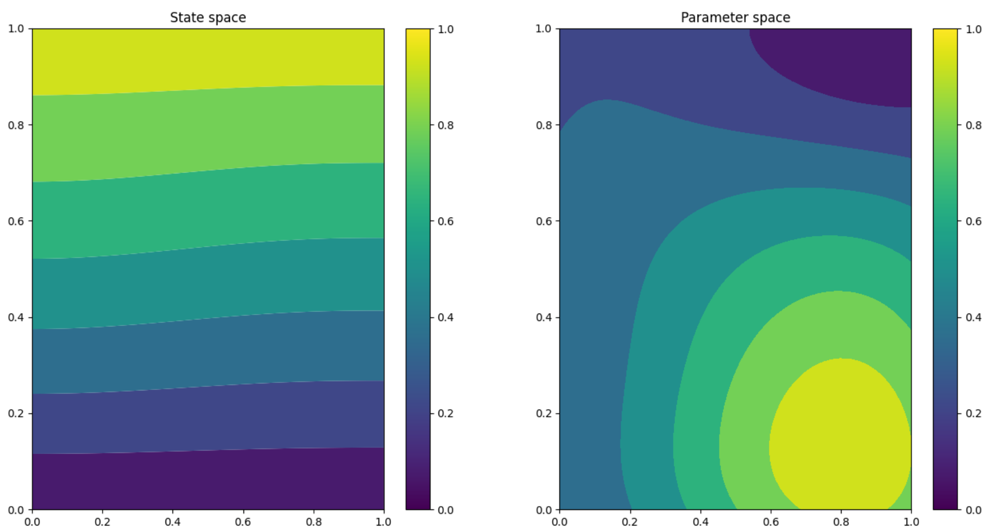

Figure 4.4: Contours of the state and parameter

Mesh independence studies revealed that the number of iterations varied between 6–7 for mesh sizes of 64, 128, and 256.

Chapter 5

Conclusion

Bibliography

[1] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[2] Martin S Alnæs, Anders Logg, Kristian B Ølgaard, Marie E Rognes, and Garth N Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 40(2):1–37, 2014.

[3] Satish Balay, Kris Buschelman, William D Gropp, Dinesh Kaushik, Matthew G Knepley, L Curfman McInnes, Barry F Smith, and Hong Zhang. Petsc. *See http://www. mcs. anl. gov/petsc*, 2001.

[4] Alfio Borzì and Volker Schulz. *Computational optimization of systems governed by partial differential equations*. SIAM, 2011.

[5] Tan Bui-Thanh and Omar Ghattas. Analysis of the hessian for inverse scattering problems: I. inverse shape scattering of acoustic waves. *Inverse Problems*, 28(5):055001, 2012.

[6] Tan Bui-Thanh, Omar Ghattas, James Martin, and Georg Stadler. A computational framework for infinite-dimensional bayesian inverse problems part i: The linearized case, with application to global seismic inversion. *SIAM Journal on Scientific Computing*, 35(6):A2494–A2523, 2013.

[7] Masoumeh Dashti and Andrew M Stuart. The bayesian approach to inverse problems. In *Handbook of uncertainty quantification*, pages 311–428. Springer, 2017.

[8] Michael Evans and Timothy Swartz. *Approximating integrals via Monte Carlo and deterministic methods*, volume 20. OUP Oxford, 2000.

[9] H Pearl Flath, Lucas C Wilcox, Volkan Akçelik, Judith Hill, Bart van Bloemen Waanders, and Omar Ghattas. Fast algorithms for bayesian uncertainty quantification in

large-scale linear inverse problems based on low-rank partial hessian approximations. *SIAM Journal on Scientific Computing*, 33(1):407–432, 2011.

[10] David Galbally, Krzysztof Fidkowski, Karen Willcox, and Omar Ghattas. Non-linear model reduction for uncertainty quantification in large-scale inverse problems. *International journal for numerical methods in engineering*, 81(12):1581–1608, 2010.

[11] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM journal on matrix analysis and applications*, 21(1):185–194, 1999.

[12] Eric Laloy, Niklas Linde, Cyprien Ruffino, Romain Hérault, Gilles Gasso, and Diederik Jacques. Gradient-based deterministic inversion of geophysical data with generative adversarial networks: is it feasible? *Computers & Geosciences*, 133:104333, 2019.

[13] Anders Logg and Garth N Wells. Dolfin: Automated finite element computing. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):1–28, 2010.

[14] Noemi Petra, Hongyu Zhu, Georg Stadler, Thomas JR Hughes, and Omar Ghattas. An inexact gauss-newton method for inversion of basal sliding and rheology parameters in a nonlinear stokes ice sheet model. *Journal of Glaciology*, 58(211):889–903, 2012.

[15] Florian Rathgeber, David A Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew TT McRae, Gheorghe-Teodor Bercea, Graham R Markall, and Paul HJ Kelly. Firedrake: automating the finite element method by composing abstractions. *ACM Transactions on Mathematical Software (TOMS)*, 43(3):1–27, 2016.

[16] Florian Rathgeber, Graham R Markall, Lawrence Mitchell, Nicolas Loriant, David A Ham, Carlo Bertolli, and Paul HJ Kelly. Pyop2: A high-level framework for performance-portable simulations on unstructured meshes. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 1116–1123. IEEE, 2012.

[17] Florian Rathgeber, Lawrence Mitchell, David Ham, Michael Lange, Andrew McRae, Fabio Luporini, Gheorghe-teodor Bercea, and Paul Kelly. Firedrake: Re-imagining fenics by composing domainspecific abstractions, 2014.

[18] Trond Steihaug. Local and superlinear convergence for truncated iterated projections methods. *Mathematical Programming*, 27:176–190, 1983.

[19] Andrew M Stuart. Inverse problems: a bayesian perspective. *Acta numerica*, 19:451–559, 2010.

[20] Luke Tierney and Joseph B Kadane. Accurate approximations for posterior moments and marginal densities. *Journal of the american statistical association*, 81(393):82–86, 1986.

[21] Andrei Nikolaevich Tikhonov. On the solution of ill-posed problems and the method of regularization. In *Doklady akademii nauk*, volume 151, pages 501–504. Russian Academy of Sciences, 1963.

[22] Fredi Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*, volume 112. American Mathematical Soc., 2010.

[23] Umberto Villa, Noemi Petra, and Omar Ghattas. hippylib: An extensible software framework for large-scale inverse problems. *Journal of Open Source Software*, 3(30), 2018.

[24] Umberto Villa, Noemi Petra, and Omar Ghattas. hIPPYlib: An Extensible Software Framework for Large-Scale Inverse Problems Governed by PDEs, 2 2020.

[25] Umberto Villa, Noemi Petra, and Omar Ghattas. hIPPYlib User Manual: Version 2. 6 2020.

[26] Curtis R. Vogel. *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics, 2002.

[27] Kainan Wang, Tan Bui-Thanh, and Omar Ghattas. A randomized maximum a posteriori method for posterior sampling of high dimensional nonlinear bayesian inverse problems. *SIAM Journal on Scientific Computing*, 40(1):A142–A171, 2018.

[28] David Williams. *Probability with martingales*. Cambridge university press, 1991.