A

# A framework for automated PDE-constrained optimisation

S. W. Funke and P. E. Farrell, Imperial College London

A generic framework for the solution of PDE-constrained optimisation problems based on the FEniCS system is presented. Its main features are an intuitive mathematical interface, a high degree of automation, and an efficient implementation of the generated adjoint model. The framework is based upon the extension of a domain-specific language for variational problems to cleanly express complex optimisation problems in a compact, high-level syntax. For example, optimisation problems constrained by the time-dependent Navier-Stokes equations can be written in tens of lines of code. Based on this high-level representation, the framework derives the associated adjoint equations in the same domain-specific language, and uses the FEniCS code generation technology to emit parallel optimised low-level C++ code for the solution of the forward and adjoint systems. The functional and gradient information so computed is then passed to the optimisation algorithm to update the parameter values. This approach works both for steady-state as well as transient, and for linear as well as nonlinear governing PDEs and a wide range of functionals and control parameters. We demonstrate the applicability and efficiency of this approach on classical textbook optimisation problems and advanced examples.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm Design and Analysis; G.1.8 [**Numerical Analysis**]: Partial Differential Equations; G.1.6 [**Numerical Analysis**]: Optimization; I.6.5 [**Simulation and Modelling**]: Model Development; J.2 [**Computer Applications**]: Physical Sciences and Engineering; J.6 [**Computer Applications**]: Computer-Aided Engineering; D.2 [**Software**]: Software Engineering

General Terms: Design, Algorithms

Additional Key Words and Phrases: optimisation, PDE constraints, adjoints, automatic differentiation, data assimilation, inverse problems

**ACM Reference Format:**
Funke, S. W. and Farrell, P. E. 2013. A framework for automated PDE-constrained optimisation. ACM Trans. Math. Softw. V, N, Article A (January YYYY), 28 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

## 1. INTRODUCTION

Optimisation problems constrained by partial differential equations (PDEs) are ubiquitous across science and engineering. Such problems consist of optimising an objective functional, e.g. maximising the performance or minimising the cost of a system, subject to constraints given by the laws of physics [Lions 1971]: for example, an aeronautical engineer will want to choose the best shape for a wing to optimise its perfor-

This work is supported by the Grantham Institute for Climate Change, a Fujitsu CASE studentship, EPSRC grant EP/I00405X/1, and a Center of Excellence grant from the Research Council of Norway to the Center for Biomedical Computing at Simula Research Laboratory. The authors would like to acknowledge helpful discussions with D. A. Ham and M. E. Rognes, the contribution of T. Surowiec to the example presented in section 6.1, and A. S. Candy and A. Avdis for the help with the mesh generation in section 6.2.
Author's address: Funke and Farrell: Department of Earth Science and Engineering, Imperial College London. Funke: Grantham Institute for Climate Change, Imperial College London. Farrell: Center for Biomedical Computing, Simula Research Laboratory, Oslo.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

mance [Jameson 1988]. Inverse problems may also be treated as optimisation problems, where the goal is to infer some unobservable state from observable evidence; this is achieved by adjusting the unknown state to minimise some misfit functional [Le Dimet and Talagrand 1986]. This approach is now fundamental to the geosciences: for example, it is routinely used in operational meteorology [Rabier et al. 2000].

Approximating the solution of PDEs is computationally expensive. This motivates the use of gradient-based optimisation algorithms, since exploring the control space without derivative information typically requires a prohibitive number of PDE evaluations for practical problems. The typical case for PDE-constrained optimisation problems is that where the dimension of the control space is large, and where the number of functionals to control is small (usually one). Therefore, their efficient solution relies on the fast gradient computation for a small number of functionals with respect to many parameters.

A naïve approach to computing the gradient of a functional is to perturb each control parameter in turn, and approximate the gradient using finite differences. A more sophisticated way would be to employ the tangent linear model associated with the forward PDE system, which circumvents the problems of roundoff errors by propagating derivative information forward through the computational graph, from one input parameter through to all outputs. However, with both of these approaches, the number of PDE solves required for a single gradient computation scales linearly with the number of parameters, making them infeasible for the typical case described above. By contrast, the adjoint method computes the gradient of a scalar functional with a single PDE solve, by propagating derivative information backwards through the computational graph, from the output functional back to all inputs [Giles and Pierce 2000; Griewank and Walther 2008]. The adjoint method is a key ingredient in making the large-scale solution of complex optimisation problems feasible.

However, deriving and implementing the adjoint PDE model is typically regarded as difficult. This has been one of the main motivations for the development of algorithmic differentiation techniques (AD, also called automatic differentiation), which attempt to automate the adjoint model derivation. However, in practice the application of an AD tool typically requires large amounts of user intervention and expertise, in particular for advanced forward model implementations. Naumann [2012, pg. *xii*] states that "the automatic generation of optimal (in terms of robustness and efficiency) adjoint versions of large-scale simulation code is one of the great open challenges in the field of High-Performance Scientific Computing". Giles and Pierce [2000] observe that

> Considering the importance of design to .. all of engineering, it is perhaps surprising that the development of adjoint codes has not been more rapid .. [I]t seems likely that part of the reason is its complexity.

In previous work, we have efficiently solved this adjoint derivation problem for the case where the forward problem may be discretised using the finite element method [Farrell et al. 2012]. The key contribution of this paper is to apply this advance to automate the solution of large classes of PDE-constrained optimisation problems. The main features of this new framework are:

*Usability.* The user specifies the discretised optimisation problem in a form that resembles the mathematical notation.

*Automation.* Based on this problem specification, the framework performs the necessary steps for the optimisation, without further user intervention. These steps include interfacing with an optimisation method, followed by repeated PDE solves for evaluating the objective functional and computing the functional gradient using the automatically derived adjoint system.

*Generality.* The framework handles large classes of governing PDEs, including coupled, nonlinear and time-dependent PDEs. Furthermore, the user may choose from multiple gradient-free and gradient-based optimisation algorithms.
*Performance.* Optimisation algorithms typically require many iterations, each of which involve computationally expensive PDE solves. For many problems of practical interest, efficient parallel PDE solvers are therefore essential to obtain reasonable run times. Significant work has been undertaken in order to produce efficient assembly code in FEniCS [Kirby and Logg 2007; Markall et al. 2012; Ølgaard and Wells 2010]. Since the adjoint model relies on the same FEniCS code generation techniques as the forward model, the performance of the adjoint implementation inherits the same efficiency and parallel scaling of the forward model. As a consequence, the achieved adjoint efficiency ratios are often close to the optimal ratio between 1 and 2; by contrast, a general AD tool typically yields efficiency ratios in the range of 3 to 30 [Naumann 2012, pg. *xi*].

These features are achieved by exploiting the particular structure of finite element models (in contrast to traditional AD, which attempts to solve the general case). Finite element discretisations of the governing PDE have a natural domain-specific language, the language of variational forms, that abstractly captures the mathematical structure of the problem without any details of how its solution is to be achieved on a particular platform. Instead of implementing the model in tens or hundreds of thousands of lines of a low-level language such as Fortran or C++, the user compactly describes the discretisation of the forward problem in the Unified Form Language (UFL) [Alnæs 2011; Alnæs et al. 2012] of the FEniCS project [Logg et al. 2012]. UFL mimics the mathematical notation almost exactly, and can express even complex time-dependent coupled nonlinear problems in tens of lines of code.

The presented framework uses the high-level UFL representation of the forward model for two purposes. First, the forward code is generated using the FEniCS project in the usual way: the UFL is passed to a dedicated finite element compiler, which *generates* optimised low-level C++ code for its parallel implementation on a particular platform [Kirby and Logg 2006; Ølgaard and Wells 2010; Markall et al. 2012]. Second, as each forward solve is executed at runtime, a symbolic tape of the forward model is recorded in UFL format. This tape is analogous to the concept of a tape in AD [Corliss and Griewank 1993], except that instead of recording individual floating-point operations, the units on the tape are whole equation solves. Once the forward model has terminated, symbolic manipulation is applied to this tape to derive the UFL representation of the associated adjoint problem, which in turn is passed to the same compiler to emit an efficient parallel implementation of the adjoint model [Farrell et al. 2012].

In this paper we discuss the extension of this system to compactly express and solve optimisation problems. The user describes the forward model, the control parameters and the objective functional in an extension of the UFL notation. The optimisation framework then repeatedly re-executes the tape to evaluate the functional value, solves the adjoint PDE to compute the functional gradient, and modifies the tape to update the position in parameter space until an optimal solution is found.

### 1.1. Related work

One of the main motivations for this work is the fact that, despite the broad applicability of PDE-constrained optimisation, there exist few software packages that gather and unify the tools required to solve such problems.

A closely related project is developed by van Bloemen Waanders et al. [2002], with the goal of creating an optimisation framework based on the finite-element software Sundance [Long et al. 2012]. Sundance is similar to FEniCS in that it also operates on

variational forms: in particular, it can automatically differentiate and adjoin individual variational forms. However, the built-in automatic adjoint derivation of Sundance does not currently extend to cases where the forward model consists of a sequence of variational problems, which is typically the case for time-dependent problems.

Other open source alternatives include DAKOTA [Eldred et al. 1996], the Stanford University Unstructured suite [Palacios et al. 2012] and RoDoBo [Becker et al. 2005]. The key difference between these and the framework presented here is that the difficult step of the adjoint derivation and implementation has not been automated. Instead, if a new PDE is to be solved the adjoint model must be derived and implemented, either manually or with the help of AD tools, both of which demand significant development effort and user expertise.

Finally, the PROPT [Rutquist and Edvall 2010], ACADO [Houska et al. 2011] and CasADi [Andersson et al. 2012] toolkits are optimisation frameworks with similar design goals, but focussed on ordinary differential equations and differential-algebraic equations instead of PDEs.

The paper is organized as follows: The next section states the general form of PDE-constrained optimisation problems and compares different solution techniques. Section 3 presents the newly developed framework in detail, followed by a code demonstration in section 4. In section 5 the framework implementation is verified using textbook examples with known analytical solutions. Finally, section 6 applies the optimisation framework to a wide range of problems before making some concluding remarks in section 7.

## 2. THE FORMULATION OF PDE-CONSTRAINED OPTIMISATION PROBLEMS

We consider the PDE-constrained optimisation problem in the following general form:

$$\begin{aligned} \min_{u,m} \; & J(u,m) \\ \text{subject to } & F(u,m) = 0, \\ & h(m) = 0, \\ & g(m) \leq 0, \end{aligned} \tag{1}$$

where the vector $m$ contains the optimisation parameters, $F(u,m) = 0$ is a system of PDEs parameterised by $m$ with solution vector $u$, and $J(u,m) \in \mathbb{R}$ is the scalar-valued objective functional that is to be minimised. The equality and inequality constraints $h(m) = 0$ and $g(m) \leq 0$ enforce additional conditions on the optimisation parameters $m$. A common example is a box constraint of the form:

$$a \leq m \leq b.$$

State constraints are not directly considered in formulation (1). However, in section 6.1 we show an example where a penalisation approach is employed to enforce state constraints.

Throughout this paper we assume that the above operators are sufficiently differentiable, and that the PDE has a unique solution for any $m$, i.e. there is a solution operator $u(m)$ such that $F(u(m), m) = 0 \quad \forall\, m$.

### 2.1. The optimality conditions and the reduced formulation

The necessary first order optimality conditions for problem (1), also known as the Karush-Kuhn-Tucker (KKT) conditions [Karush 1939; Kuhn and Tucker 1951], are derived from the associated Lagrangian. Ignoring the control constraints for simplicity, the Lagrangian of (1) is:

$$\mathcal{L}(u,m,\lambda) \equiv J(u,m) + \lambda^T F(u,m) \in \mathbb{R},$$

where $\lambda$ is the Lagrange multiplier (also known as the dual or adjoint variable). The KKT conditions state that for every local minimum $(\bar{u}, \bar{m})$ at which some regularity conditions are satisfied (see Hinze et al. [2009] for details), there exists a Lagrange multiplier $\bar{\lambda}$ such that:

$$\frac{\partial \mathcal{L}}{\partial u}(\bar{u}, \bar{m}) = \frac{\partial J}{\partial u}(\bar{u}, \bar{m}) + \bar{\lambda}^T \frac{\partial F}{\partial u}(\bar{u}, \bar{m}) = 0, \tag{2a}$$

$$\frac{\partial \mathcal{L}}{\partial m}(\bar{u}, \bar{m}) = \frac{\partial J}{\partial m}(\bar{u}, \bar{m}) + \bar{\lambda}^T \frac{\partial F}{\partial m}(\bar{u}, \bar{m}) = 0, \tag{2b}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda}(\bar{u}, \bar{m}) = F(\bar{u}, \bar{m}) = 0. \tag{2c}$$

Equation (2b) is referred to as the adjoint equation; (2c) recovers the governing PDE. Solving (2b) for $\bar{\lambda}$ and substituting the result into the control equation (2a) yields that the total derivative of the objective functional vanishes at the optimal point.

One approach to compute a local solution of problem (1), known as the all-at-once approach, simultaneous analysis and design, or the oneshot approach, is to directly solve the KKT system (2). A common solution method is sequential quadratic programming (SQP), which for the simplified case considered here is equivalent to applying Newton's method to the KKT system (2). A key advantage of SQP is that it inherits the fast local quadratic convergence to a local solution from Newton's method [Boggs and Tolle 1995]. However, the KKT system (2) yields significant challenges for numerical solvers: it is a coupled, nonlinear and often ill-conditioned system of PDEs and the resulting linear systems that need to be solved for each SQP update are high-dimensional. This issue becomes particularly problematic in the case of time-dependent governing PDEs where the discrete solution vectors $u$ and $\lambda$ contain the entire forward and adjoint solution trajectories over time and space.

Since direct solver methods are typically not suitable to solve linear problems of such dimensions, iterative solvers in combination with advanced preconditioning techniques are often applied and show promising results in certain applications [Battermann and Sachs 2001; Biros and Ghattas 2000; Schöberl and Zulehner 2007]. An alternative approach is to use a space reduction method; here, the solution of the full linear systems is avoided by performing a block-LU decomposition [Byrd and Nocedal 1990; Biegler et al. 1995; Schulz 1998]. As a result, the system is uncoupled and can be solved in separate steps of more manageable size. Reduced SQP methods have been successfully applied to various applications [Kupfer and Sachs 1992; Orozco and Ghattas 1992; Orozco and Ghattas 1997].

The all-at-once approach has the disadvantage that the current estimate of $u$, $m$ and $\lambda$ must be stored at any time. For large, time-dependent problems, storing the whole estimate of the forward and adjoint solutions $u$ and $\lambda$ can quickly exceed the available memory: for example, a simulation with $10^6$ spatial degrees of freedom and $10^8$ timesteps would require roughly $1000$ TB of memory in double point precision, exceeding the memory capacities of most available computers.

This issue can be circumvented by performing a space reduction on the original optimisation problem (1). This approach (also known as black-box or nested analysis and design approach) replaces the objective functional $J(u, m)$ with the reduced functional $\hat{J}(m) \equiv J(u(m), m)$, that is the functional is considered as a pure function of the optimisation parameters. Since the reduced functional implicitly enforces the solution of the PDE, the PDE-constraint becomes superficial in the optimisation formulation.

The result is the following reduced optimisation problem:

$$\begin{aligned} \min_m \hat{J}(m) \\ \text{subject to } h(m) = 0, \\ g(m) \leq 0. \end{aligned} \tag{3}$$

This formulation has the practical advantage that the dimension of the optimisation problem is greatly reduced, since the dimension of $m$ is typically much smaller than that of the PDE solution $u$. Consequently, many robust and established optimisation methods are directly applicable. Furthermore, the storage requirement is significantly lowered; firstly because the adjoint solution is only used for computing the functional gradient and does not need to be saved and secondly because the storage of the entire forward solution trajectory may be avoided by using a checkpointing strategy to balance storage and computation cost. This allows the solution of large scale optimisation problems for which storing the whole forward solution would be impossible [Griewank 1992]. In the example above, the optimal checkpointing scheme implemented by Griewank and Walther [2000] with $1000$ checkpoints reduces the storage cost to approximately $10$ GB while the computational cost approximately doubles. Another advantage of the reduced formulation is that the governing PDE is always satisfied after each optimisation iteration. Hence, the optimisation loop may be stopped as soon as the functional is sufficiently reduced, simplifying the formulation of termination criteria.

Long et al. [2012] state that for steady problems, the all-at-once can outperform the reduced formulation by a wide margin, but that for time-dependent systems, the reduced formulation is often preferable. Since the framework supports time-dependent problems, the current implementation solves the optimisation problem in the reduced formulation (3). However, in principle all components for solving the all-at-once approach are available, and we plan to implement this approach in future work. Furthermore, the reduced formulation is often used to precondition the all-at-once approach [Biros and Ghattas 2000].

The following example illustrates the two formulations (1) and (3) on a classical optimal control problem (see for example Tröltzsch [2005, chapter 2.1.5] or Hinze et al. [2009, chapter 1.5.3]): given a thin, heatable plate $\Omega \subset \mathbb{R}^2$ with fixed temperature at the domain boundary $\partial\Omega$, what is the optimal heating function that should be applied to obtain a desired temperature profile? This problem can be formulated as an optimisation problem constrained by the stationary heat equation:

$$\begin{aligned} \min_{u,m} \ & \frac{1}{2}\|u - u_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|m\|^2_{L^2(\Omega)} \\ \text{subject to } & -\nabla^2 u = m && \text{on } \Omega, \\ & u = 0 && \text{on } \partial\Omega, \\ & a \leq m \leq b && \text{on } \Omega. \end{aligned} \tag{4}$$

Here $u_d : \Omega \to \mathbb{R}$ is the desired temperature profile and $u : \Omega \to \mathbb{R}$ is the solution of the stationary heat equation with homogeneous Dirichlet boundary conditions. The objective functional measures the misfit between the PDE solution and the desired temperature profile plus a regularisation term that is multiplied by a scaling factor $\alpha \geq 0$. The optimisation parameter $m : \Omega \to \mathbb{R}$ controls the heat source and is limited by the box constraints.

Under mild assumptions, the heat equation with Dirichlet boundary conditions yields a unique solution for any source parameter $m$ [Hinze et al. 2009, §1.3.1.1]. Hence

there exists a solution operator $u(m)$ and the reduced problem may be formulated:

$$\min_m \ \frac{1}{2}\|u(m) - u_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|m\|^2_{L^2(\Omega)}$$
$$\text{subject to } a \leq m \leq b \qquad \text{on } \Omega.$$

This problem will be used for the code example in section 4.

## 3. THE OPTIMISATION FRAMEWORK

The core of the framework relies on two software components: first, the FEniCS system [Logg et al. 2012; Logg 2007] is used to solve the forward and adjoint PDEs. Second, it relies on libadjoint and dolfin-adjoint [Farrell et al. 2012] to automatically derive the associated adjoint system for gradient computations. In this work we have extended the dolfin-adjoint framework to go beyond adjoint derivation, to automate the solution of PDE-constrained optimisation problems.

The following sections discuss these components in detail. The source code, including the examples and applications in the following sections, is available at `http://dolfin-adjoint.org`.

### 3.1. The FEniCS system

The FEniCS system is a collection of software components for automating the solution of PDEs by the finite element method. This section gives a brief introduction to the FEniCS system. A thorough overview can be found in Logg et al. [2012].

To solve a PDE with the FEniCS system, the user defines its discretised weak form in the domain specific language UFL that mimics and encodes the mathematical formulation [Alnæs 2011; Alnæs et al. 2012]. This high-level formulation is then passed to a finite element form compiler such as FFC [Kirby and Logg 2006], which generates optimised low-level code for the evaluation of the local element tensors. This generated code is used by DOLFIN [Logg and Wells 2010; Logg et al. 2011] to globally assemble and solve the problem. DOLFIN also provides the data structures for meshes, function spaces and functions.

For time-dependent PDEs, the temporal discretisation is usually performed with a non-finite element discretisation scheme. In this case, the user writes the time loop manually and solves the variational problem for each timelevel as described above.

DOLFIN has interfaces for both C++ and Python. The Python interface uses just in time compilation, i.e. it invokes the necessary compilers at runtime. In contrast, the code generation for the C++ interface happens at a preprocessing step before running the forward model. As a consequence, the high-level description of the forward problem is not directly available at runtime in the C++ interface. Because dolfin-adjoint relies on this data to perform runtime inspection and manipulation, the remaining sections discuss only the Python interface to DOLFIN.

### 3.2. libadjoint and dolfin-adjoint

The libraries libadjoint and dolfin-adjoint enable the automatic derivation and solution of tangent linear and adjoint models from forward models written in DOLFIN.

The purpose of libadjoint is to facilitate the development of tangent linear and adjoint models based on the fundamental abstraction of considering the forward model as a sequence of equation solves. Based on this abstraction, the library builds a symbolic description of the forward model, the tape, from which it can automatically derive the symbolic representation of the associated tangent linear and adjoint systems.

The software library dolfin-adjoint acts as the between DOLFIN and libadjoint. It inspects DOLFIN's problem description at runtime, and performs the required tasks

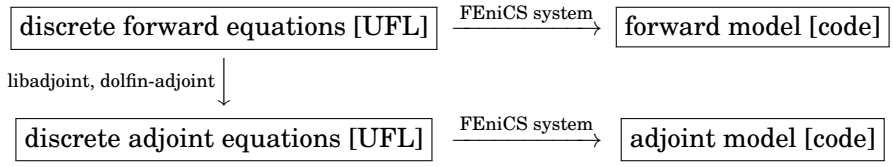



Fig. 1: The automated generation of the adjoint model with dolfin-adjoint. The user specifies the discrete forward equations in the high-level UFL language. From that symbolic representation of the problem, libadjoint and dolfin-adjoint can derive the corresponding representation of the discrete adjoint equations in UFL. Both the forward and adjoint equations are passed to the FEniCS system to generate the forward and adjoint model implementations.

for applying libadjoint without user intervention. The tangent linear and adjoint models produced with libadjoint are represented in the same high-level data format as the forward model. Therefore, the code generation techniques in FEniCS can be applied to the adjoint model in the same manner as the forward model, see figure 1. Farrell et al. [2012] showed that this approach leads to a robust, automatic and efficient way of implementing tangent linear and adjoint models.

### 3.3. The optimisation framework

*3.3.1. Introduction.* The proposed framework extends dolfin-adjoint to solve PDE-constrained optimisation problems. The optimisation process consists of iteratively evaluating the functional of interest and its gradient at different points in the parameter space. The key idea is to automate these evaluations by operating purely on the tape of the forward model recorded by libadjoint. In particular, a functional evaluation is obtained by replaying the tape (which runs the forward model) while simultaneously evaluating the objective functional. The functional gradient is computed by deriving and solving the adjoint model, as described in the previous section. When the optimisation algorithm updates the point in parameter space, the tape is modified accordingly.

With this approach, the only inputs required from the user are the objective functional, the control parameters and the governing PDEs, optionally with additional equality and inequality constraints.

*3.3.2. User interface.* The solver for an optimisation problem is typically implemented in the following three steps.

First, the user implements the governing PDE in the Python interface of DOLFIN [Logg et al. 2012]. DOLFIN supports linear and nonlinear as well as steady and transient PDEs, and has been used to solve complex coupled systems such as the Reynolds-averaged Navier-Stokes equations for turbulent flows [Mortensen et al. 2011], the Stokes equations for mantle convection [Vynnytska et al. 2013], and the Cahn-Hilliard equations for phase separation [Wells et al. 2006].

Second, the user defines the objective functional and the optimisation parameters. For that, a `Functional` class has been introduced that extends DOLFIN to support the expression of time-dependent functionals. For example, an objective functional computing:

$$\int_0^T \int_\Omega u \cdot v \, \mathrm{d}x \, \mathrm{d}t$$

is defined by:

```
J = Functional(inner(u, v)*dx*dt)
```

Alternatively, the functional can be evaluated at a specific time. For example

$$\int_\Omega u(t=1)\cdot v(t=1)\,\mathrm{d}x,$$

is implemented by:

```
J = Functional(inner(u, v)*dx*dt[1])
```

Functionals may consist of forms integrated over time, forms evaluated at a particular time, and sums of these. This allows the construction of complex objective functionals, e.g. a data assimilation problem with a regularisation term involving the initial condition might use the following objective functional:

$$\int_0^T \|u-u_{\text{obs}}\|^2_{L^2(\Omega)}\,\mathrm{d}t + |u(t=0)|^2_{H^1(\Omega)}.$$

The implementation of that functional is:

```
J = Functional(inner(u - u_obs, u - u_obs)*dx*dt
             + inner(grad(u), grad(u))*dx*dt[0])
```

Similarly, the user specifies the optimisation parameter $m$. For example, the following code defines the optimisation parameter to be the initial value of $u$:

```
m = InitialConditionParameter(u)
```

If a scalar value $s$ is to be optimised, one may use

```
m = ScalarParameter(s)
```

In the final step, the user defines the reduced functional $\hat{J}$ with

```
J_hat = ReducedFunctional(J, m)
```

In order to optimise for multiple parameters simultaneously, the user can optionally pass a list of `Parameter` objects.

At this point, the reduced functional object `J_hat` can be used to solve the associated minimisation problem $\min_m \hat{J}(m)$ by calling:

```
m_opt = minimize(J_hat)
```

or the associated maximisation problem $\max_m \hat{J}(m)$ by calling:

```
m_opt = maximize(J_hat)
```

Both of these functions solve the optimisation problem with the default settings and return the optimised parameters when finished. Additional arguments may be used to set and configure the optimisation algorithm and to define box, inequality or equality constraints, e.g.:

```
m_opt = maximize(J_hat, bounds = (u_lower, u_upper), method='SLSQP')
```

Currently, the framework supports most of the algorithms in the optimisation package of SciPy [Jones et al. 2001]. For problems without additional constraints these are:

— the Nelder-Mead method [Nelder and Mead 1965];
— a modification to Powell's method [Powell 1964];
— a nonlinear conjugate gradient method [Nocedal and Wright 2006, §5.2];
— the BFGS method [Nocedal and Wright 2006, §6.1];
— the Newton-CG method [Nocedal and Wright 2006, §7.1]
— and simulated annealing [Laarhoven and Aarts 1987].

Both Powell's method and simulated annealing are gradient-free optimisation algorithms. For problems with box, inequality or equality constraints, the user has the choice between:

— sequential quadratic programming [Kraft 1994];
— the gradient-free Constrained Optimization by Linear Approximation method [Powell 1994].

Finally, if only box-constraints are present, the

— L-BFGS-B method [Nocedal and Wright 2006, §7.2];
— a Newton-CG implementation that supports box constraints [Nash 1984],

may be used.

The user also has access to more advanced features, such as to automatically verify each gradient computation with the Taylor remainder convergence test during the optimisation procedure, or to execute user-supplied code after each gradient computation, for example to create convergence plots.

*3.3.3. Implementation.* The `ReducedFunctional` class provides two main functionalities: the evaluation of the objective functional for a given parameter value, and the computation of the functional gradient.

The functional evaluation is implemented by solving the forward equations that have been stored on libadjoint's tape and simultaneously computing the objective functional. However, the tape contains the parameter values of the initial forward run and a naïve replay would reevaluate the forward model with the original parameters. This issue is resolved by first modifying the tape so that it reflects the most recent parameter values before executing the functional evaluation.

The implementation of the gradient computation relies on the adjoint derivation of libadjoint and dolfin-adjoint to compute the gradient with the adjoint approach. Optionally, the user may use a checkpointing scheme to balance the storage and recomputation cost of the gradient computation [Griewank 1992; Farrell et al. 2012].

The `minimize` and `maximize` routines implement the interface to the optimisation algorithms. Optimisation methods typically require the implementation of the functional evaluation and its gradient. The `minimize` and `maximize` routines generate these functions using the `ReducedFunctional` object. This involves the conversion of DOLFIN data structures to generic array data types on which the optimisation algorithms operate.

Parallel execution is often crucial in PDE-constrained optimisation to achieve reasonable run times. While DOLFIN supports the parallel solution of the forward and adjoint models, the considered implementations of the optimisation algorithms are not designed for distributed execution. The `minimize` and `maximize` routines circumvent this problem by executing the PDE solves in parallel, but replicating the optimisation computation on each processor. For that, the `minimize` and `maximize` functions serialise the distributed data structures of the optimisation parameters, the functional gradient and the constraints, as these are used by the optimisation algorithm. All other variables, in particular the forward and adjoint solutions, remain distributed and are not communicated. This approach works well for small-scale optimisation problems,

where the communication time for gathering the data and the execution time spent in the optimisation algorithm is small compared to the runtime of the PDE solves. For large-scale optimisation problems one should consider interfacing a parallel optimisation algorithm, such as TAO [Munson et al. 2012] or OPT++ [Meza 1994].

Finally, there are cases where optimisation based purely on libadjoint's tape is not desired. For example, a forward model with an adaptive timestepping scheme changes the timestep according to certain conditions. This adaptivity is not reflected in the tape, and hence the optimisation process would only use the timestep choice that was used to build the tape. In such cases, the user can manually implement the functional evaluation; however, this approach has the disadvantage that the computed gradients become inconsistent with the discrete forward model, which can result in a reduced convergence of the optimisation algorithm.

### 3.4. Restrictions

The first restrictions are those associated with the adjoint computation [Farrell et al. 2012, §5.4]. In particular, the automated adjoint derivation relies on the differentiability of the forward model, and that the forward model is implemented entirely in the Python interface to DOLFIN.

State constraints other than the PDE constraint are not handled automatically. However, the framework does provide many of the tools required to implement the solution of such problems. An example is presented in section 6.1, where a problem with a variational inequality is broken down into a sequence of PDE-constrained optimisation problems, each of which is solved with the framework presented here.

Another restriction is that shape optimisation is not yet automated. It is possible to apply the shape calculus approach [Schmidt and Schulz 2010; Schmidt et al. 2011] which derives an expression for the shape gradient of the functional in terms of the forward and adjoint solutions, which may then be computed using the automatically generated adjoint model. In future work, we plan to automate this shape calculus analysis to extend the framework to support automated shape optimisation.

## 4. EXAMPLE CODE

This section demonstrates the application of the presented optimisation framework to two optimal control problems. The first example solves the optimal heating problem (4). The governing PDE in this case is the stationary heat equation. The second example replaces the stationary heat equation with a time-dependent, nonlinear PDE. Although this problem adds significant complexity to the forward and adjoint PDEs, the required code changes are minimal.

### 4.1. Distributed control of the heat equation

The first example solves the optimal control problem of the stationary heat equation (4). The problem possesses a unique optimal solution if the box constraints $a$ and $b$ are bounded [Tröltzsch 2005, chapter 2.5.1]. Otherwise the regularisation term must be strictly positive, i.e. $\alpha > 0$, to ensure uniqueness. In the following example, these conditions are satisfied by choosing $\alpha = 0$, $a = 0$ and $b = 0.5$.

To begin with, the user implements and solves the forward problem. By doing so, the optimisation framework creates the tape of the forward model. This tape is used by the optimisation framework to compute all future functional evaluations and gradient computations.

For this example the two-dimensional domain $\Omega \equiv [-1, 1]^2$ was uniformly discretised using triangular elements. The finite-dimensional function spaces are constructed using $\mathrm{P1}_{\mathrm{CG}}$ elements for the PDE solution $u$ and the desired temperature profile $u_d$, and $\mathrm{P0}_{\mathrm{DG}}$ elements for the heat source $m$. The latter choice is motivated by the fact that

the control can possess discontinuities for $\alpha = 0$. The following Python code initialises the domain, the function spaces and all required functions in DOLFIN:

```
# Define the domain [-1, 1] x [-1, 1]
n = 200
mesh = RectangleMesh(-1, -1, 1, 1, n, n)
# Define the function spaces
V = FunctionSpace(mesh, "CG", degree = 1)
W = FunctionSpace(mesh, "DG", degree = 0)
# Define the functions
u = Function(V, name = "Solution")
m = Function(W, name = "Control")
v = TestFunction(V)
```

The weak form of the heat equation is obtained by multiplying the PDE with a test function $v \in V$, then integrating the result over the domain and integrating by parts. The resulting weak formulation is: find $u \in V$ such that:

$$\langle \nabla u, \nabla v \rangle_\Omega = \langle m, v \rangle_\Omega \quad \forall v \in V.$$

The associated code resembles the mathematical notation closely:

```
# Solve the forward model to create the tape
F = (inner(grad(u), grad(v)) - m*v)*dx
bc = DirichletBC(V, 0.0, "on_boundary")
solve(F == 0, u, bc)
```

Next, the objective functional is defined. In this example, the desired temperature profile $u_d$ is defined as:

$$u_d(x, y) = e^{-1/(1-x^2)-1/(1-y^2)},$$

plotted in figure 2a. The relevant code is:

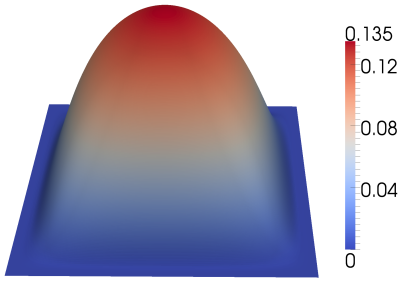
```
# Define the functional of interest
u_d = exp(-1/(1-x[0]*x[0])-1/(1-x[1]*x[1]))
J = Functional((0.5*inner(u-u_d, u-u_d))*dx)
```

At this point, the optimal control problem can be solved:

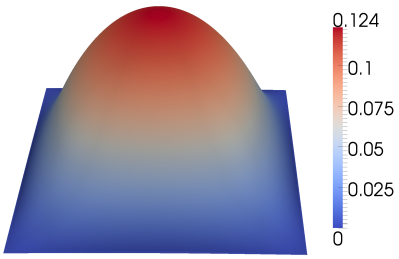
```
# Define the reduced functional
J_hat = ReducedFunctional(J, SteadyParameter(m))
# Solve the optimisation problem
m_opt = minimize(J_hat, method = "L-BFGS-B", bounds = (0.0, 0.5),
                 options = {"gtol": 1e-16, "ftol": 1e-16})
```

The last parameter of `minimize` sets the termination condition to stop if either the infinity norm of the projected gradient or the relative change of the functional value drops below the specified tolerance.

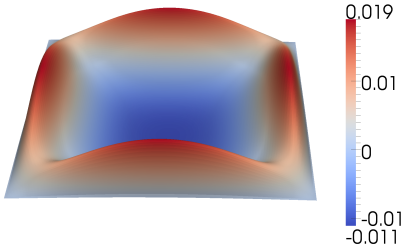
The optimisation algorithm starts with a zero estimate for the control, i.e. $m^{(0)} = 0$. The corresponding functional value is $\hat{J}\left(m^{(0)}\right) = 8.9 \times 10^{-3}$. The results of the optimisation are shown in figure 2. The final value of the objective functional is $1.4 \times 10^{-4}$. The desired and optimised temperature profile are of similar shape, but their maximum values differ significantly. This is reflected in the fact that the box constraints are active in large parts of the domain (figure 2d).



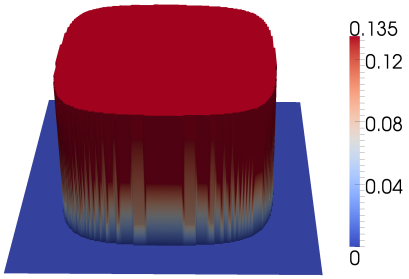

(a) Desired temperature profile $u_d$




(b) Optimised temperature profile $u$




(c) Difference between optimised and desired temperature profiles $u - u_d$




(d) Optimised heat source control $m$

Fig. 2: The solutions of the stationary optimal heating problem with box constraints. The heat source control is limited by the box constraints in large parts of the domain, which leads to a relatively large difference between the desired and optimised temperature profiles.

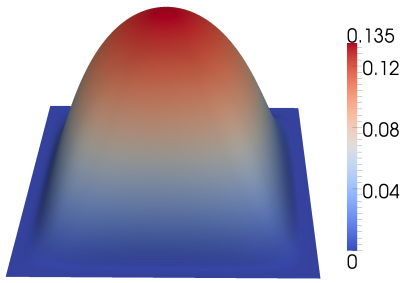
By removing the box constraints, a better agreement between the desired and optimised temperature profiles is expected. Therefore the box constraints were removed and the regularisation parameter set to $\alpha = 10^{-7}$, in order to ensure the uniqueness of the optimal solution. The results are shown in figure 3. Compared to the previous results, the pointwise difference between the desired and optimised temperature profiles is significantly decreased, yielding a final functional is $1.6 \times 10^{-7}$.

### 4.2. Distributed control of a nonlinear, time-dependent PDE

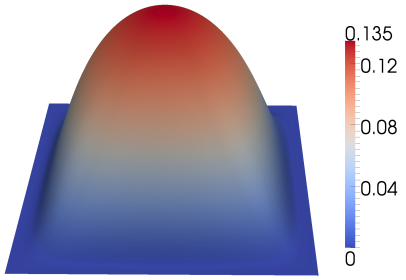
This example modifies the optimal heating problem by replacing the stationary heat equation with a time-dependent, nonlinear PDE:

$$\begin{aligned} \min_{m} \ & \frac{1}{2}\|u(t=T) - u_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|m\|^2_{L^2(\Omega)} \\ \text{subject to } & \frac{\partial u}{\partial t} - \nabla^2 u + u^3 = m && \text{on } \Omega \times (0,T], \\ & u = 0 && \text{on } \partial\Omega \times (0,T], \\ & u = 0 && \text{for } \Omega \times \{0\}, \\ & a \leq m \leq b && \text{on } \partial\Omega \times (0,T], \end{aligned} \tag{5}$$

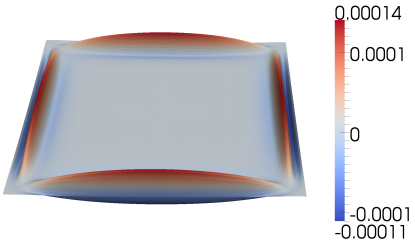
where $u : \Omega \times (0,T] \to \mathbb{R}$ and $u_d : \Omega \times (0,T] \to \mathbb{R}$ are the time-dependent PDE solution and desired temperature profile, respectively, and the control $m : \Omega \to \mathbb{R}$ is constant



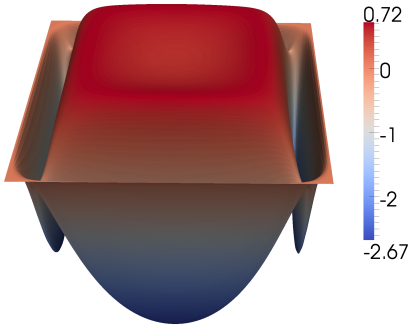

(a) Desired temperature profile $u_d$




(b) Optimised temperature profile $u$




(c) Difference between optimal and desired temperature profiles $u - u_d$




(d) Optimised heat source control $m$

Fig. 3: The solutions of the stationary optimal heating problem without box constraints. The optimised heat source control achieves a good agreement between the desired and optimised temperature profiles.

in time. The nonlinearity and time-dependency of the new governing PDE adds significant complexity to the solution process of the optimisation algorithm. In particular, the gradient computations involve the storage of the forward solution trajectory (or the application of some checkpointing scheme) and the solution of the associated time-dependent adjoint system.

However, since the steps for recording the tape and the functional and gradient evaluations are automated, the code from the previous example can be reused almost entirely to solve problem (5). The only modification required is to replace the weak formulation of the heat equation with the new governing PDE. The following Python code shows an example implementation with a backward Euler time discretisation and a Newton solver for the nonlinear equation solve at each timestep:

```
# Define the weak form
timestep = 0.01
F = ((u - u_old)*v/timestep + inner(grad(u), grad(v)) + u**3*v - m*v)*dx
# Perform 10 timesteps
for t in range(11):
    solve(F == 0, u, bc)
    u_old.assign(u)
    adj_inc_timestep()
```

| Element size | $\|m - m^{\text{opt}}\|$ | order | $\|u - u^{\text{opt}}\|$ | order |
|---|---|---|---|---|
| $1.25 \times 10^{-1}$ | $9.54 \times 10^{-2}$ | | $3.83 \times 10^{-4}$ | |
| $6.25 \times 10^{-2}$ | $3.70 \times 10^{-2}$ | 1.34 | $9.03 \times 10^{-5}$ | 2.08 |
| $3.12 \times 10^{-2}$ | $1.71 \times 10^{-2}$ | 1.11 | $2.20 \times 10^{-5}$ | 2.03 |
| $1.56 \times 10^{-2}$ | $8.33 \times 10^{-3}$ | 1.04 | $5.43 \times 10^{-6}$ | 2.02 |
| $7.81 \times 10^{-3}$ | $4.11 \times 10^{-3}$ | 1.02 | $1.36 \times 10^{-6}$ | 2.00 |

Table I: The rate of convergence for the smooth control test. The control error shows the expected first order convergence, and the PDE solution converges as expected at second order.

The optimisation framework can also use a checkpointing scheme to reduce the storage cost of the gradient computations. For example, the multistage checkpointing scheme developed by Stumm and Walther [2009], which supports storing checkpoints both in RAM and on disk, is activated with:

```
adj_checkpointing(strategy='multistage', steps=11,
                  snaps_on_disk=2, snaps_in_ram=3)
```

## 5. VERIFICATION

This section applies the optimisation framework to problems with analytical solutions in order to compare the numerical order of convergence with the theoretical expectation. An agreement of the convergence rates is considered to be a strong indicator that the implementation is correct [Salari and Knupp 2000].

The considered analytical solutions are based on the optimal control problem (4) of the heat equation, extended with an additional source term $s : \Omega \to \mathbb{R}$:

$$\begin{aligned} \min_{m} \ & \|u - u_d\|^2_{L^2(\Omega)} & \\ \text{subject to } & -\nabla^2 u = m + s & \text{in } \Omega, \\ & u = 0 & \text{on } \partial\Omega, \\ & -1 \leq m \leq 1 & \text{in } \Omega. \end{aligned} \tag{6}$$

The following sections perform the two convergence tests, the first one with a continuous optimal control function, and the second one with a discontinuous optimal control function.

### 5.1. Smooth control

The first test is based on an analytical solution with a smooth optimal control function:

$$\begin{aligned} m^{\text{opt}}(x, y) &\equiv \sin(\pi x) \sin(\pi y), \\ u^{\text{opt}}(x, y) &\equiv u_d^{\text{opt}}(x, y) \equiv \frac{1}{2\pi^2} \sin(\pi x) \sin(\pi y), \\ s &\equiv 0. \end{aligned}$$

It is easy to see that this choice forms an optimal solution to problem (6).

The discretisation and optimisation parameters are configured identically to the setup described in section 4.1. Then the convergence test was performed on five uniformly discretised meshes with decreasing mesh element sizes. The resulting errors and convergence rates are given in table I. The first-order convergence for the control solution $m$ is expected as the underlying function space is discretised with $\text{P0}_{\text{DG}}$ finite elements. Similarly, a second-order rate of convergence is observed for the PDE solution $u$, as it is discretised with $\text{P1}_{\text{CG}}$ finite elements.

| Element size | $\|m - m^{\text{opt}}\|$ | order | $\|u - u^{\text{opt}}\|$ | order |
|---|---|---|---|---|
| $3.13 \times 10^{-2}$ | $4.76 \times 10^{-2}$ | | $7.38 \times 10^{-4}$ | |
| $1.56 \times 10^{-2}$ | $2.41 \times 10^{-2}$ | 0.99 | $2.01 \times 10^{-4}$ | 1.89 |
| $7.81 \times 10^{-3}$ | $1.20 \times 10^{-2}$ | 1.00 | $5.04 \times 10^{-5}$ | 1.99 |
| $3.91 \times 10^{-3}$ | $5.30 \times 10^{-3}$ | 1.18 | $1.24 \times 10^{-5}$ | 2.03 |
| $1.95 \times 10^{-3}$ | $2.30 \times 10^{-3}$ | 1.20 | $2.54 \times 10^{-6}$ | 2.29 |

Table II: The rate of convergence for the bang-bang control test. The control error shows the expected first order convergence, and the PDE solution converges at second order as expected.

### 5.2. Bang-bang control

The second verification test is motivated by the fact that box constraints can lead to optimal control solutions with discontinuities. The following test, derived in Tröltzsch [2005, chapter 2.9.1], yields an optimal control function with discontinuities in a chessboard-like shape:

$$m^{\text{opt}}(x, y) \equiv -\text{sign}\left(-\sin\left(8\pi x\right)\sin\left(8\pi y\right)\right).$$

This kind of control, where the control values jump from one box constraint limit to the other, is also known as bang-bang control. The optimal state solution is chosen to be:

$$u^{\text{opt}}(x, y) \equiv \sin(\pi x)\sin(\pi y).$$

By applying the optimality conditions, the source term $s$ and the desired PDE solution $u_d$ are obtained:

$$s(x, y) \equiv 2\pi^2 \sin(\pi x)\sin(\pi y) + \text{sign}\left(-\sin\left(8\pi x\right)\sin\left(8\pi y\right)\right),$$

and:

$$u_d(x, y) \equiv \sin(\pi x)\sin(\pi y) + \text{sign}\left(-\sin\left(8\pi x\right)\sin\left(8\pi y\right)\right).$$

The convergence test is performed with the same configuration as in the previous test. The resulting errors and convergence rates are given in table II. It shows the expected first-order convergence for the control and second-order convergence for the state solution, indicating that the optimisation implementation is correct.

## 6. APPLICATIONS

### 6.1. Optimal control governed by an elliptic variational inequality

In this section we apply the framework to an optimal control problem investigated in Hintermüller and Kopacka [2011, §5.2]. This problem involves a variational inequality constraint on the state, and is an example of a mathematical program with equilibrium constraints (MPEC). Let $K = \{v \in H_0^1(\Omega) : v \geq 0\}$. The problem is stated as:

$$\min_{u,m} \; J(u, m) = \frac{1}{2}\left|\left|u - u_d\right|\right|^2_{L^2(\Omega)} + \frac{\nu}{2}\left|\left|m\right|\right|^2_{L^2(\Omega)} \tag{7a}$$

$$\text{subject to} \;\; u \in K, \tag{7b}$$

$$\langle \nabla u, \nabla(v - u)\rangle_\Omega \geq \langle f + m, v - u\rangle_\Omega \;\; \forall\, v \in K, \tag{7c}$$

$$a \leq m \leq b \;\; \text{a.e. in } \Omega, \tag{7d}$$

where $u : \Omega \to \mathbb{R}^2$ is the state variable, $u_d : \Omega \to \mathbb{R}^2$ is a prescribed state to be matched, $m : \Omega \to \mathbb{R}$ is the control variable to be determined, $f : \Omega \to \mathbb{R}$ is a prescribed source term, $a \in \mathbb{R}$ and $b \in \mathbb{R}$ are the lower and upper bounds on the control, and $\nu \in \mathbb{R}$ is a

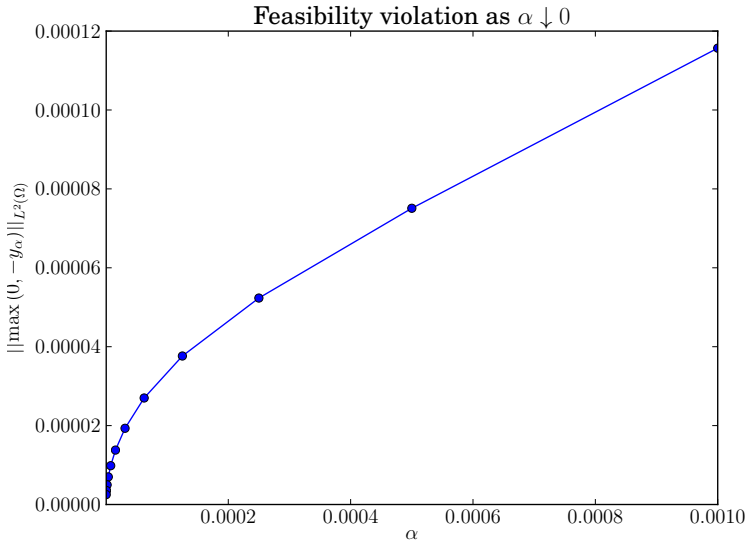



Fig. 4: The feasibility violation of the state solution $u$ as a function of $\alpha$. As $\alpha$ approaches 0, the variational inequality $u \geq 0$ a.e. in $\Omega \subset \mathbb{R}^2$ is enforced.

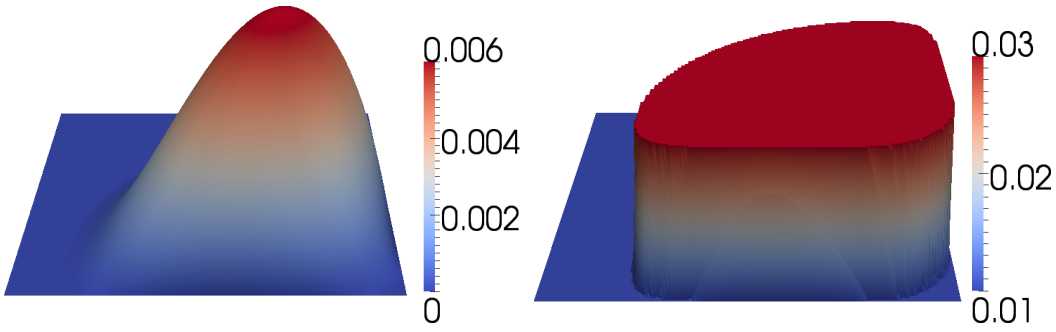
regularisation parameter. Note that the inequality $u \geq 0$ *a.e.* is implied by $u \in K$. Following the penalisation approach [Trémolières et al. 1981; Hintermüller and Kopacka 2011], the variational inequality can be approximated by the penalised equation:

$$(\nabla u, \nabla v) + \frac{1}{\alpha}(-\max(0, -u), v) = (f + m, v) \quad \forall\, v \in H_0^1(\Omega), \tag{8}$$

where $\alpha > 0$ is the penalty parameter. It is well known that the solution $u_\alpha$ of (8) converges to that of the variational inequality (7c) as $\alpha \downarrow 0$. As the $\max$ operator is not differentiable, it is regularised in turn with a smoothing parameter $\epsilon > 0$ (the "global" regularisation of Hintermüller and Kopacka [2011, equation (2.4)]). Therefore, to solve (7), a sequence of problems are solved where the variational inequality (7c) is replaced with the regularised penalised equality constraint (8), and the penalisation parameter $\alpha$ is driven to zero. The solution of one iteration is used as the initial guess for the next.

The PDE constraint is discretised using linear finite elements for the state and control. The PDE is first solved once with a zero control $m$ to build a tape of the forward model, and then all subsequent steps are performed by operating on this tape. The value of $\epsilon$ is set to $10^{-4}$, while $\nu$ is set to $10^{-2}$. The remaining parameters are taken from Hintermüller and Kopacka [2011, §5.2]). The value of $\alpha$ is initialised to $10^{-3}$ and halved at each penalisation iteration. The penalised subproblem is solved with a call to `minimize`, which applies a limited-memory BFGS algorithm with control bounds support [Zhu et al. 1997]. The entire program consists of less than 50 lines of code.

The feasibility of the state is measured by computing the diagnostic $||\max(0, -u)||_{L^2(\Omega)}$; figure 4 shows its evolution as a function of $\alpha$. The control and state solutions of the optimal control problem are shown in figure 5. Excellent agreement is found with the solutions of Hintermüller and Kopacka [2011, figures 5 and 6], giving confidence that the solutions are correct.




(a) The optimised state $u$ (b) The optimised control $m$

Fig. 5: The solutions of the optimal control problem with a variational inequality.

| | Runtime (s) | Ratio |
|---|---|---|
| Forward model | 69.08 | |
| Forward model + adjoint model | 77.85 | 1.126 |

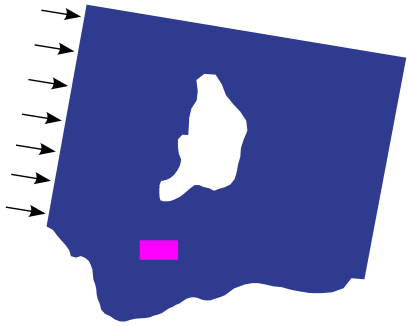
Table III: Timings for the MPEC gradient calculation. The efficiency of the adjoint approaches the theoretical ideal value of 1.125.

Finally, the efficiency of the gradient calculation $\nabla \hat{J}$ is benchmarked. For gradient-based optimisation algorithms to be practical, the computation of the gradient must be affordable. To investigate the performance of the adjoint-based gradient calculation, one execution of the forward and adjoint models was timed; this was repeated several times to ensure the results were representative. The results are shown in table III. As the forward model in this case takes eight Newton iterations to converge, the adjoint should take approximately one eighth the cost of the forward model, for an ideal $R = (\text{forward} + \text{adjoint})/(\text{forward})$ ratio of $1.125$. The observed ratio is $1.126$, demonstrating the claim in Farrell et al. [2012] that the gradient calculation with dolfin-adjoint approaches optimal theoretical efficiency.

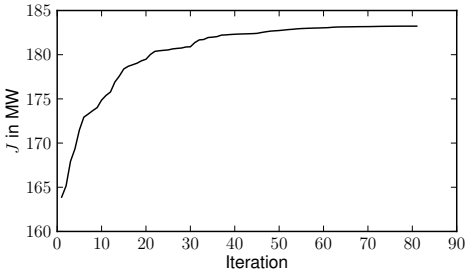
### 6.2. Optimal placement of tidal turbines

This application investigates an essential problem in the tidal energy industry. The core idea is to place turbines in the ocean to extract the kinetic energy of the tidal flow and convert it into electricity. In order to extract an economically useful amount of power, many turbines (possibly hundreds) must be deployed in the tidal stream. The question is: how should these turbines be placed in relation to each other to maximise the power extracted? The strong nonlinear interaction between the turbines, the complicated constraints on the configuration (legal site restrictions, bounds on the gradient of the seafloor), and the sensitive dependence of the power on the configuration make it difficult to manually identify an optimal configuration.

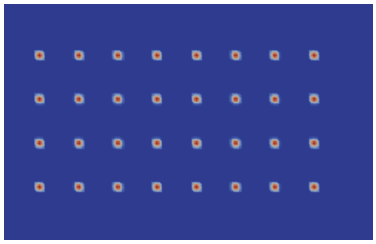
This problem is formulated as an optimisation problem constrained by the stationary, nonlinear shallow water equations with appropriate initial and boundary condi-



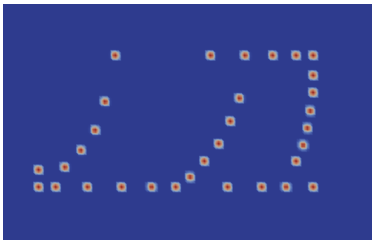
(a) The computational domain; the feasible turbine positions are coloured pink




(b) The objective functional



(c) Initial turbine positions



(d) Optimised turbine positions

Fig. 6: The solution of the optimal turbine layout problem.

tions:

$$\begin{aligned} &\max_{u,m} \; J(u,m) \\ &\text{subject to } u\cdot\nabla u - \nu\nabla^2 u + g\nabla\eta = \frac{c_b + c_t(m)}{H}\|u\|u \quad \text{on } \Omega, \\ &\qquad\qquad \nabla\cdot(Hu) = 0 \qquad\qquad\qquad\qquad\qquad\quad \text{on } \Omega, \end{aligned} \tag{9}$$

where $\Omega \subset \mathbb{R}^2$ is the computational domain, the unknowns $u : \Omega \to \mathbb{R}^2$ and $\eta : \Omega \to \mathbb{R}$ are the depth-averaged velocity and the free-surface displacement, respectively, $H \in \mathbb{R}$ is the water depth at rest, $g \in \mathbb{R}$ is the gravitational force, $\nu \in \mathbb{R}$ is the viscosity coefficient, and $c_b \in \mathbb{R}$ and $c_t$ represent the quadratic bottom friction and the turbine parameterisation, respectively. In a practical application, problem formulation (9) should of course be extended to the time-dependent shallow water equations to account for the flood and ebb tides.

The functional of interest $J$ is defined to be the power extracted due to the increased friction in the turbine farm [Ben Elghali et al. 2007; Divett et al. 2011]:

$$J(u,m) \equiv \frac{1}{2}\int_\Omega \rho c_t(m)\|u\|^3 \mathrm{d}\Omega, \tag{10}$$

where $\rho$ is the density of water.

The vector of parameters $m \in \mathbb{R}^{2N}$ in (9) encodes the $x$ and $y$ positions of $N$ turbines as:

$$m = (p_1^x, p_1^y, p_2^x, p_2^y, \ldots, p_N^x, p_N^y)^T.$$

| | Runtime (s) | Ratio |
|---|---|---|
| Forward model | 30.41 | |
| Forward model + adjoint model | 34.22 | 1.125 |

Table IV: Timings for the turbine position gradient calculation. The efficiency of the adjoint is that of the theoretical value of 1.125.

The turbines are parameterised by increased friction located around the turbine centres. The corresponding friction function $c_t(m)$ is defined as:

$$c_t(m)(x,y) \equiv \sum_{i=1}^{N} K\psi_{p_i^x,r}(x)\psi_{p_i^y,r}(y), \tag{11}$$

where $K = 21$ is a scaling factor, $r = 40$ m is the extent of the parameterised turbine, and $\psi$ is a smooth bump function with compact support defined as:

$$\psi_{p,r}(x,y) \equiv \begin{cases} e^{1-1/(1-\|\frac{(x,y)^T-p}{r}\|^2)} & \text{for } \|\frac{(x,y)^T-p}{r}\| < 1, \\ 0 & \text{otherwise.} \end{cases}$$

The shallow water equations were discretised using the P2-P1 finite-element pair. For performance reasons, the function $c_t(m)$ was implemented in Python instead of expressing it as part of the UFL formulation. Consequently, the dependency on $m$ does not occur explicitly in the UFL form and hence dolfin-adjoint cannot automatically compute its derivative. However, dolfin-adjoint is able to automatically compute the derivative of $J$ with respect to $c_t$, and so this problem can be circumvented by overloading the `ReducedFunctional` class and manually implementing the final step of the chain rule:

$$\nabla \hat{J}(m) = \frac{\mathrm{d}\hat{J}}{\mathrm{d}c_t}\frac{\mathrm{d}c_t}{\mathrm{d}m}.$$

The first term is automatically computed using dolfin-adjoint. The second term can easily be derived and implemented by hand by differentiating (11). Once this `ReducedFunctional` class was implemented, the optimisation framework could be used as usual.

The example considered here optimises a deployment site near the Orkney Islands in Scotland, where 32 turbines are to be installed (figure 6a). A constant input flow with 2 m/s speed is enforced on the left boundary, while the free-surface displacement on the right boundary is set to zero. A no-normal flow condition is applied on all remaining boundaries. The remaining parameters are $H = 50$ m, $\nu = 90$ m$^2$/s, $g = 9.81$ m/s$^2$.

The turbines are initially distributed in a structured manner as shown in figure 6c. The optimisation was performed using the SQP implementation of Kraft [1994] until the relative reduction of the functional of interest dropped below $10^{-6}$. Bound constraints ensured that the turbines remained in the site area, which models the fact that site developers acquire a license for a particular site, and cannot deploy outside it. Furthermore, a set of inequality constraints were used to enforce a minimum distance of 60 m between each turbine.

The results are presented in figure 6. The optimisation algorithm terminated after 81 iterations. The optimisation increased the total power output of the turbine farm by 12%, from an initial value of 164 MW to 183 MW.

Table IV compares the runtime of the forward model and the runtime of the gradient calculation. Both were performed in parallel with 4 CPUs. The ratio of forward and adjoint runtimes is close to the theoretical ideal, as expected.

### 6.3. Data assimilation with wetting and drying

*6.3.1. Introduction.* Wetting and drying processes such as tsunami inundation or the flooding and receding of tides play an important role in the study of tsunamis [Kowalik et al. 2005], tidal flats and river estuaries [Zhang et al. 2009; Xue and Du 2010], and flooding events [Westerink et al. 2008; Song et al. 2011]. Many algorithms have been proposed for the numerical simulation of wetting and drying processes, both for the shallow water equations (Medeiros and Hagen [2013] and the references therein) and for the Navier-Stokes equations [Funke et al. 2011].

In this example, we consider a data assimilation problem where the goal is to reconstruct the profile of an incoming tsunami from observations of the wet/dry inundation interface. The tsunami is modelled by the time-dependent nonlinear shallow water equations with the wetting and drying scheme proposed by Kärnä et al. [2011]. The resulting optimisation problem is:

$$\begin{aligned} &\min_{m,u,\eta} \; J(\eta) \\ &\text{subject to } \frac{\partial u}{\partial t} + (u\cdot\nabla)u + g\nabla\eta = \frac{c_t(\tilde{H})}{\tilde{H}}\|u\|u \quad \text{on } \Omega\times(0,T], \\ &\qquad\qquad \frac{\partial \tilde{H}}{\partial t} + \nabla\cdot(\tilde{H}u) = 0 \qquad\qquad\qquad\quad \text{on } \Omega\times(0,T], \\ &\qquad\qquad \tilde{H} = m \qquad\qquad\qquad\qquad\qquad\quad \text{on } \partial\Omega_D\times(0,T]. \end{aligned} \tag{12}$$

where $\Omega \subset \mathbb{R}^2$ is the spatial domain, $T$ is the final time and $u : \Omega\times(0,T] \to \mathbb{R}^2$ and $\eta : \Omega\times(0,T] \to \mathbb{R}$ are the unknown depth-averaged velocity and free-surface displacement, respectively. In the classical shallow water equations the total water depth is defined as $H \equiv \eta + h$, where $h : \Omega \to \mathbb{R}$ is the static bathymetry; in order to account for wetting and drying, Kärnä et al. [2011] uses a modified total depth definition $\tilde{H} \equiv f(H)$ instead, where $f$ is a smooth approximation of the maximum operator:

$$f(H) \equiv \frac{1}{2}\left(\sqrt{H^2+\alpha^2}+H\right) \approx \max(0,H),$$

and $\alpha > 0$ controls the accuracy of the approximation. The remaining parameters in (12) are the gravitational force $g = 9.81\,\mathrm{m/s^2}$ and the friction coefficient in the Chézy-Manning formulation:

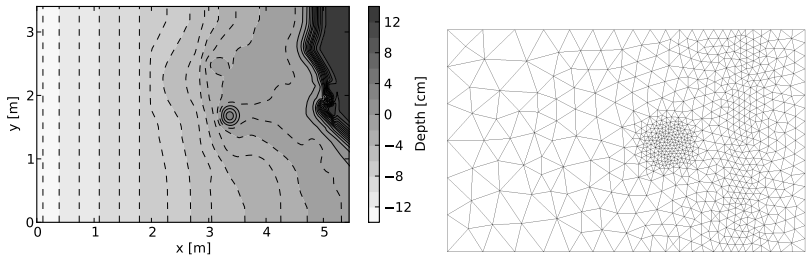
$$c_t(\tilde{H}) = \frac{g\mu^2}{\tilde{H}^{1/3}},$$

where $\mu \in \mathbb{R}$ is the user specified Manning coefficient.

The boundary conditions are as follows: on the inflow boundary $\partial\Omega_D$ a Dirichlet boundary condition with value $m : (0,T] \to \mathbb{R}$ is applied, which also acts as the control parameter. For simplicity, it is assumed that $m$ varies only in time, i.e. is constant along the boundary. On the remaining boundaries, a no-normal flow boundary condition is applied.

The functional of interest $J$ measures the misfit between the observed and the simulated wet/dry interface. For its formulation, an indicator function is constructed that is $1$ in dry areas and $0$ in wet areas. By noting that $\eta \geq h$ in wet areas and $\eta < h$ in dry areas, this indicator function is defined as $\mathcal{H}(\eta - h)$ where $\mathcal{H}$ denotes the following smooth approximation of the Heaviside step function:

$$\mathcal{H}(x) \equiv \frac{1}{2}\left(\frac{x}{\sqrt{x^2+\alpha^2}}+1\right) \approx \begin{cases} 0 & \text{if } x<0, \\ 1 & \text{else}, \end{cases}$$




(a) Bathymetry (b) Mesh

Fig. 7: The laboratory setup of the Hokkaido-Nansei-Oki tsunami example, based on the $1{:}400$ laboratory experiment of Matsuyama and Tanaka [2001]. The island at the center and the coast on the right are hit by a tsunami coming from the left boundary.

where again $\alpha$ controls the smoothness of the approximation. With that, the functional of interest is defined as:

$$J(\eta) \equiv \frac{1}{2}\int_0^T ||\mathcal{H}(\eta - h) - d||^2_{L^2(\Omega)}\ \mathrm{d}t,$$

where $d : \Omega \times (0, T] \to \mathbb{R}$ denotes the indicator function of the observed wet/dry interface. While such inverse problems are in general ill-posed and require regularisation, satisfactory numerical results were obtained in this example with no regularisation term. The implementation of such a regularisation term in the functional would be a trivial modification.

*6.3.2. Implementation.* The modified shallow water equations in the optimisation problem (12) are discretised with the LBB-stable $\mathrm{P1_{DG}}$-P2 finite element pair in space [Cotter et al. 2009]. A simple upwinding scheme is implemented, which is obtained by integrating the advection term by parts, replacing the advected velocity at the inflow facets with the upwind velocity and then integrating by parts again. Following Kärnä et al. [2011], the resulting equations are then discretised with a second-order Diagonally Implicit Runge-Kutta (DIRK22) scheme in time [Ascher et al. 1997, §2.6].

The implementation of problem (12) in the presented optimisation framework was straightforward: the control parameters appear directly in the UFL representation of the governing equations, and hence the framework was applicable without any modifications.

*6.3.3. Reconstruction of the Hokkaido-Nansei-Oki tsunami profile.* This example is motivated by the question of whether it is possible to reconstruct a tsunami profile from satellite observations that record the inundation interface on the coast over time.

The considered event is the Hokkaido-Nansei-Oki tsunami that occurred in 1993 and produced run-up heights of up to $30$ m on Okushiri island, Japan. The Central Research Institute for Electric Power Industry (CRIEPI) in Abiko, Japan constructed a $1{:}400$ laboratory scale model of the area around the island [Matsuyama and Tanaka 2001]. Following the setup used in Yalciner et al. [2011], we consider a rectangular domain of size $5.448$ m$\times 3.402$ m, with the bathymetry and coastal topography shown in figure 7a. It contains an island in the center and coastal regions on the top right of the domain. The left boundary is the inflow boundary $\partial\Omega_D$, on which a surface elevation profile is enforced that resembles a tsunami (figure 8a). The task is to reconstruct this wave profile.

The domain is discretised with an unstructured mesh consisting of $1,411$ triangular elements with increasing resolution near the inundation areas (figure 7b). The tem-

| | Runtime (s) | Ratio |
|---|---|---|
| Forward model | 2026 | |
| Forward model + adjoint model | 3146 | 1.55 |

Table V: Timings for the tsunami reconstruction gradient calculation. The efficiency of the adjoint approaches the theoretical ideal value of 1.5.

poral discretisation uses a timestep of $0.5$ s with a total simulation time of $32$ s. The Manning coefficient was set to $\mu = 0.05\ \mathrm{s/m}^{\frac{1}{3}}$ and a smoothness value of $\alpha = 0.1$ was used.

The observations $d$ are synthetically generated by running the forward model with the wave profile that was used in the laboratory experiment while recording the wet/dry interface. This approach of generating the synthetic observation data with the same model that is used for the assimilation is referred to as an "inverse crime" [Kaipio and Somersalo 2004], which renders the optimisation problem less ill-posed than it actually is. However, the main purpose of this example is to demonstrate the capabilities of the optimisation framework, and hence this approach is adopted for simplicity.

The optimisation algorithm begins with an initial guess for the Dirichlet boundary values of $0.105$ cm for all timelevels, which corresponds to the final free-surface displacement of the input wave. The tsunami signal at the boundary is applied $2$ s after the simulation start time. The boundary condition during the final $2$ s has no impact on the functional, as the wave does not affect the wet/dry interface before the simulation ends. Therefore, the boundary values at the start and end were reset to the correct Dirichlet boundary values and excluded from the optimisation. Furthermore, a box constraint was used to restrict the minimum and maximum free-surface displacement between $-1.5$ cm and $+2$ cm; without these constraints the first optimisation iterations generate unrealistically large Dirichlet boundary values for which the forward model does not converge.

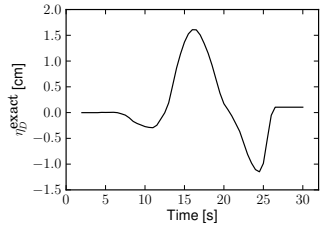
Figure 8 shows the results of the problem solved with the limited memory BFGS (L-BFGS-B) implementation in SciPy [Jones et al. 2001]. After $103$ optimisation iterations ($113$ functional evaluations) the relative decrease of the functional of interest fell below machine precision and the algorithm terminated. The incoming wave was reconstructed to within an absolute error of $3.91 \times 10^{-7}$ cm (figure 8c), which corresponds to a relative error of less than $3 \times 10^{-5}\%$ of the incoming wave height.

Table V lists the runtimes of the forward model and the gradient calculation. The nonlinear equations of the forward model are typically solved with 2 Newton iterations, which suggests an optimal runtime ratio of 1.5. The measured value is close to this theoretical value and confirms the relative efficiency of the adjoint model implementation.

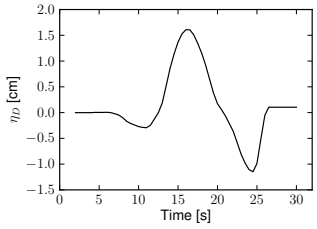
## 7. SUMMARY

In this paper we present a new framework for rapidly defining and solving PDE-constrained optimisation problems. The framework exploits the FEniCS system, dolfin-adjoint, and established optimisation algorithms to allow the user to specify the discretised optimisation problem in a high-level language that resembles the mathematical notation.

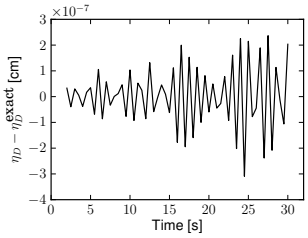
The core idea of the implementation is to perform all required tasks of the optimisation process using the tape of the forward model that is recorded by dolfin-adjoint (analogous to the AD concept of a tape). This includes the evaluation of the objective functional by replaying the forward model, computing the functional gradient by deriving and solving the adjoint problem, and modifying the tape in order to incorporate



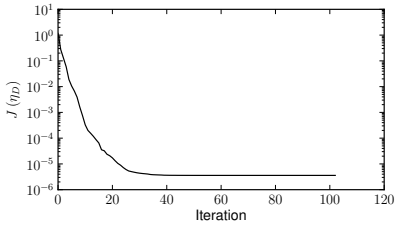

(a) The input tsunami profile $m^{\text{opt}}$




(b) The reconstructed tsunami profile $m$




(c) The difference between input and reconstructed tsunami profiles $\left|m - m^{\text{opt}}\right|$




(d) The objective functional $J$

Fig. 8: Results of the reconstruction of the Hokkaido-Nansei-Oki tsunami profile. The initial and final 2 s of the boundary values are excluded from the reconstruction.

parameter updates. While this paper applies the idea to the dolfin-adjoint system, the same approach is applicable to the operator-overloading class of AD tools that build a tape at runtime.

As demonstrated, this approach reduces the required user input to a minimum: once the forward model has been implemented, only a handful of lines of code are required to specify the optimisation problem. It applies naturally to both linear and nonlinear as well as to both steady and time-dependent governing PDEs. Furthermore, the user has the choice of a variety of gradient-free and gradient-based methods. General equality and inequality control constraints can be applied.

In this paper, the reduced formulation is used for solving the optimisation problem. For cases where quasi-Newton methods applied to the reduced formulation are insufficient, the framework provides all ingredients necessary for more sophisticated approaches. Therefore, this framework is also of interest for the development of such advanced optimisation algorithms: by implementing an algorithm in the framework once, it is immediately applicable to optimisation problems across science and engineering. Future work includes the automation of shape optimisation, the development of the oneshot approach, multigrid optimisation techniques, and the exploitation of reduced-order modelling in optimisation.

## REFERENCES

ALNÆS, M. S. 2011. UFL: A finite element form language. In *Automated Solution of Differential Equations by the Finite Element Method*, A. Logg, K. A. Mardal, and G. N. Wells, Eds. Springer-Verlag, Berlin, Heidelberg, New York, Chapter 17, 299–334. 3, 7

ALNÆS, M. S., LOGG, A., ØLGAARD, K. B., ROGNES, M. E., AND WELLS, G. N. 2012. Unified Form Language: A domain-specific language for weak formulations of partial differential equations. arXiv:1211.4047 [cs.MS]. 3, 7

ANDERSSON, J., ÅKESSON, J., AND DIEHL, M. 2012. CasADi – a symbolic package for automatic differentiation and optimal control. In *Recent Advances in Algorithmic Differentiation*, S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds. Lecture Notes in Computational Science and Engineering Series, vol. 87. Springer-Verlag, Berlin, Heidelberg, New York. 4

ASCHER, U. M., RUUTH, S. J., AND SPITERI, R. J. 1997. Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics 25,* 2-3, 151–167. 22

BATTERMANN, A. AND SACHS, E. W. 2001. Block preconditioners for KKT systems in PDE-governed optimal control problems. In Fast solution of discretized optimization problems, K.-H. Hoffmann, R. H. W. Hoppe, and V. Schulz, Eds. *International Series Of Numerical Mathematics 138*, 1–18. 5

BECKER, R., MEIDNER, D., AND VEXLER, B. 2005. RODOBO: A C++ library for optimization with stationary and nonstationary PDEs. 4

BEN ELGHALI, S. E., BALME, R., LE SAUX, K., EL HACHEMI BENBOUZID, M., CHARPENTIER, J. F., AND HAUVILLE, F. 2007. A simulation model for the evaluation of the electrical power potential harnessed by a marine current turbine. *IEEE Journal of Oceanic Engineering 32,* 4, 786–797. 19

BIEGLER, L. T., NOCEDAL, J., AND SCHMID, C. 1995. A reduced Hessian method for large-scale constrained optimization. *SIAM Journal on Optimization 5,* 2, 314–347. 5

BIROS, G. AND GHATTAS, O. 2000. Parallel Lagrange-Newton-Krylov-Schur methods for PDE-constrained optimization. Part I: The Krylov-Schur solver. *SIAM Journal on Scientific Computing 27*, 2005. 5, 6

BOGGS, P. T. AND TOLLE, J. W. 1995. Sequential quadratic programming. *Acta Numerica 4,* 1, 1–51. 5

BYRD, R. H. AND NOCEDAL, J. 1990. An analysis of reduced Hessian methods for constrained optimization. *Mathematical Programming 49,* 1–3, 285–323. 5

CORLISS, G. F. AND GRIEWANK, A. 1993. Operator overloading as an enabling technology for automatic differentiation. In *Proceedings of the Annual Object-Oriented Numerics Conference*. Sunriver, Oregon. 3

COTTER, C. J., HAM, D. A., AND PAIN, C. C. 2009. A mixed discontinuous/continuous finite element pair for shallow-water ocean modelling. *Ocean Modelling 26,* 1-2, 86–90. 22

DIVETT, T., VENNELL, R., AND STEVENS, C. 2011. Optimisation of multiple turbine arrays in a channel with tidally reversing flow by numerical modelling with adaptive mesh. In *Proceedings of the European Wave and Tidal Energy Conference 2011*. Southampton, UK. 19

ELDRED, M. S., OUTKA, D. E., BOHNHOFF, W. J., WITKOWSKI, W. R., ROMERO, V. J., PONSLET, E. R., AND CHEN, K. S. 1996. Optimization of complex mechanics simulations with object-oriented software design. *Computer Modeling and Simulation in Engineering 1*, 323–352. 4

FARRELL, P. E., HAM, D. A., FUNKE, S. W., AND ROGNES, M. E. 2012. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*. Accepted. 2, 3, 7, 10, 11

FUNKE, S. W., PAIN, C. C., KRAMER, S. C., AND PIGGOTT, M. D. 2011. A wetting and drying algorithm with a combined pressure/free-surface formulation for non-hydrostatic models. *Advances in Water Resources 34,* 11, 1483–1495. 21

GILES, M. B. AND PIERCE, N. A. 2000. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion 65,* 3-4, 393–415. 2

GRIEWANK, A. 1992. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software 1,* 1, 35–54. 6, 10

GRIEWANK, A. AND WALTHER, A. 2000. Algorithm 799: revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software 26,* 1, 19–45.

GRIEWANK, A. AND WALTHER, A. 2008. *Evaluating derivatives: Principles and techniques of algorithmic differentiation* Second Ed. SIAM, Philadelphia. 2

HINTERMÜLLER, M. AND KOPACKA, I. 2011. A smooth penalty approach and a nonlinear multigrid algorithm for elliptic MPECs. *Computational Optimization and Applications 50,* 1, 111–145. 17

HINZE, M., PINNAU, R., ULBRICH, M., AND ULBRICH, S. 2009. *Optimization with PDE constraints*. Mathematical Modelling: Theory and Applications Series, vol. 23. Springer-Verlag, Berlin, Heidelberg, New York. 6

HOUSKA, B., FERREAU, H. J., AND DIEHL, M. 2011. ACADO toolkit – an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods 32,* 3, 298–312. 4

JAMESON, A. 1988. Aerodynamic design via control theory. *Journal of Scientific Computing 3,* 3, 233–260. 2

JONES, E., OLIPHANT, T., PETERSON, P., ET AL. 2001. SciPy: Open source scientific tools for Python. 9, 23

KAIPIO, J. AND SOMERSALO, E. 2004. *Statistical and computational inverse problems*. Applied Mathematical Sciences Series, vol. 160. Springer-Verlag, Berlin, Heidelberg, New York. 23

KÄRNÄ, T., DE BRYE, B., GOURGUE, O., LAMBRECHTS, J., COMBLEN, R., LEGAT, V., AND DELEERSNIJDER, E. 2011. A fully implicit wetting–drying method for DG-FEM shallow water models, with an application to the Scheldt estuary. *Computer Methods in Applied Mechanics and Engineering 200,* 5–8, 509–524.

KARUSH, W. 1939. Minima of functions of several variables with inequalities as side constraints. M.S. thesis, Department of Mathematics, University of Chicago. 4

KIRBY, R. C. AND LOGG, A. 2006. A compiler for variational forms. *ACM Transactions on Mathematical Software 32,* 3, 417–444. 3, 7

KIRBY, R. C. AND LOGG, A. 2007. Efficient compilation of a class of variational forms. *ACM Transactions on Mathematical Software 33,* 3, 93–138. 3

KOWALIK, Z., KNIGHT, W., LOGAN, T., AND WHITMORE, P. 2005. Numerical modeling of the global tsunami: Indonesian tsunami of 26 December 2004. *Science of Tsunami Hazards 23,* 1, 40– 56. 21

KRAFT, D. 1994. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software 20,* 3, 262–281. 10

KUHN, H. W. AND TUCKER, A. W. 1951. Nonlinear programming. In *Second Berkeley symposium on mathematical statistics and probability*. Berkeley, California, 481–492. 4

KUPFER, F. S. AND SACHS, E. W. 1992. Numerical solution of a nonlinear parabolic control problem by a reduced SQP method. *Computational Optimization and Applications 1,* 1, 113–135. 5

LAARHOVEN, P. J. M. AND AARTS, E. H. L., Eds. 1987. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell. 10

LE DIMET, F. X. AND TALAGRAND, O. 1986. Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects. *Tellus A 38A,* 2, 97–110. 2

LIONS, J. L. 1971. *Optimal control of systems governed by partial differential equations*. Grundlehren der mathematischen Wissenschaften Series, vol. 170. Springer-Verlag, Berlin, Heidelberg, New York. 1

LOGG, A. 2007. Automating the finite element method. *Archives of Computational Methods in Engineering 14,* 2, 93–138. 7

LOGG, A., MARDAL, K.-A., AND WELLS, G. N., Eds. 2012. *Automated Solution of Differential Equations by the Finite Element Method*. Lecture Notes in Computational Science and Engineering Series, vol. 84. Springer-Verlag, Berlin, Heidelberg, New-York. 3, 7, 8

LOGG, A. AND WELLS, G. N. 2010. DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software 37,* 2. 7

LOGG, A., WELLS, G. N., AND HAKE, J. E. 2011. DOLFIN: A C++/Python finite element library. In *Automated Solution of Differential Equations by the Finite Element Method*, A. Logg, K.-A. Mardal, and G. N. Wells, Eds. Springer-Verlag, Berlin, Heidelberg, New-York, Chapter 10. 7

LONG, K., BOGGS, P. T., AND VAN BLOEMEN WAANDERS, B. G. 2012. Sundance: high-level software for PDE-constrained optimization. *Scientific Programming 20,* 3, 293–310. 3

MARKALL, G. R., SLEMMER, A., HAM, D. A., KELLY, P. H. J., CANTWELL, C. D., AND SHERWIN, S. J. 2012. Finite element assembly strategies on multi-core and many-core architectures. *International Journal for Numerical Methods in Fluids 71,* 1, 80–97. 3

MATSUYAMA, M. AND TANAKA, H. 2001. An experimental study of the highest runup height in the 1993 Hokkaido Nansei-oki earthquake tsunami. In *National Tsunami Hazard Mitigation Program Review and International Tsunami Symposium (ITS 2001)*. 879–889. 22

MEDEIROS, S. C. AND HAGEN, S. C. 2013. Review of wetting and drying algorithms for numerical tidal flow models. *International Journal for Numerical Methods in Fluids 71,* 4, 473–487.

MEZA, J. C. 1994. OPT++: An object-oriented class library for nonlinear optimization. Tech. Rep. SAND94-8225, Sandia National Laboratory, California, USA. 11

MORTENSEN, M., LANGTANGEN, H. P., AND WELLS, G. N. 2011. A FEniCS-based programming framework for modeling turbulent flow by the Reynolds-averaged Navier–Stokes equations. *Advances in Water Resources 34,* 9, 1082–1101. 8

MUNSON, T., SARICH, J., WILD, S., BENSON, S., AND MCINNES, L. C. 2012. TAO 2.1 users manual. Tech. Rep. ANL/MCS-TM-322, Argonne National Laboratory, Illinois, USA. 11

NASH, S. 1984. Newton-type minimization via the Lanczos method. *SIAM Journal on Numerical Analysis 21,* 4, 770–788. 10

NAUMANN, U. 2012. *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. Software, Environments, and Tools Series, vol. 24. SIAM, Philadelphia. 3

NELDER, J. A. AND MEAD, R. 1965. A simplex method for function minimization. *The Computer Journal 7,* 4, 308–313. 10

NOCEDAL, J. AND WRIGHT, S. J. 2006. *Numerical optimization* 2 Ed. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, Berlin, Heidelberg, New-York. 10

ØLGAARD, K. B. AND WELLS, G. N. 2010. Optimizations for quadrature representations of finite element tensors through automated code generation. *ACM Transactions on Mathematical Software 37,* 1, 8–1. 3

OROZCO, C. E. AND GHATTAS, O. N. 1992. Massively parallel aerodynamic shape optimization. *Computing Systems in Engineering 3,* 1-4, 311–320. 5

OROZCO, C. E. AND GHATTAS, O. N. 1997. A reduced SAND method for optimal design of non-linear structures. *International Journal for Numerical Methods in Engineering 40,* 15, 2759–2774. 5

PALACIOS, F., ALONSO, J., COLONNO, M., HICKEN, J., AND LUKACZYK, T. 2012. Adjoint-based method for supersonic aircraft design using equivalent area distributions. In *50th AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, Nashville, Tennessee. 4

POWELL, M. J. D. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal 7,* 2, 155–162. 10

POWELL, M. J. D. 1994. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Proceedings of the Sixth Workshop on optimization and numerical analysis*, S. Gomez and J. P. Hennart, Eds. Oaxaca, Mexico, 51–67. 10

RABIER, F., JÄRVINEN, H., KLINKER, E., MAHFOUF, J. F., AND SIMMONS, A. 2000. The ECMWF operational implementation of four-dimensional variational assimilation. I: Experimental results with simplified physics. *Quarterly Journal of the Royal Meteorological Society 126,* 564, 1143–1170. 2

RUTQUIST, P. E. AND EDVALL, M. M. 2010. *PROPT Matlab Optimal Control Software*. 4

SALARI, K. AND KNUPP, P. 2000. Code verification by the method of manufactured solutions. Tech. Rep. SAND2000-1444, Sandia National Laboratories, New Mexico, USA. 15

SCHMIDT, S., ILIC, C., SCHULZ, V., AND GAUGER, N. R. 2011. Airfoil design for compressible inviscid flow based on shape calculus. *Optimization and Engineering 12*, 349–369. 11

SCHMIDT, S. AND SCHULZ, V. 2010. Shape derivatives for general objective functions and the incompressible Navier-Stokes equations. *Control and Cybernetics 39,* 3, 677–713. 11

SCHÖBERL, J. AND ZULEHNER, W. 2007. Symmetric indefinite preconditioners for saddle point problems with applications to PDE-constrained optimization problems. *SIAM Journal on Matrix Analysis and Applications 29,* 3, 752–773. 5

SCHULZ, V. H. 1998. Solving discretized optimization problems by partially reduced SQP methods. *Computing and Visualization in Science 1,* 2, 83–96. 5

SONG, L., ZHOU, J., LI, Q., YANG, X., AND ZHANG, Y. 2011. An unstructured finite volume model for dam-break floods with wet/dry fronts over complex topography. *International Journal for Numerical Methods in Fluids 67,* 8, 960–980. 21

STUMM, P. AND WALTHER, A. 2009. Multistage approaches for optimal offline checkpointing. *SIAM Journal on Scientific Computing 31,* 3, 1946–1967.

TRÉMOLIÈRES, R., LIONS, J.-L., AND GLOWINSKI, R. 1981. *Numerical analysis of variational inequalities*. Studies in Mathematics and its Applications Series, vol. 8. North-Holland, Amsterdam, New-York, Oxford. 17

TRÖLTZSCH, F. 2005. *Optimale Steuerung partieller Differentialgleichungen - Theorie, Verfahren und Anwendungen* 2 Ed. Vieweg+Teubner, Wiesbaden. 11

VAN BLOEMEN WAANDERS, B., BARTLETT, R., LONG, K., BOGGS, P., AND SALINGER, A. 2002. Large scale non-linear programming for PDE constrained optimization. Tech. Rep. SAND2002-3198, Sandia National Laboratories, New Mexico, California, USA.

VYNNYTSKA, L., ROGNES, M. E., AND CLARK, S. R. 2013. Benchmarking FEniCS for mantle convection simulations. *Computers & Geosciences 50,* 0, 95–105. 8

WELLS, G. N., KUHL, E., AND GARIKIPATI, K. 2006. A discontinuous Galerkin method for the Cahn–Hilliard equation. *Journal of Computational Physics 218,* 2, 860–877. 8

WESTERINK, J. J., LUETTICH, R. A., FEYEN, J. C., ATKINSON, J. H., DAWSON, C., ROBERTS, H. J., POWELL, M. D., DUNION, J. P., KUBATKO, E. J., AND POURTAHERI, H. 2008. A basin- to channel-scale unstructured grid hurricane storm surge model applied to southern Louisiana. *Monthly Weather Review 136,* 3, 833–864. 21

XUE, H. AND DU, Y. 2010. Implementation of a wetting-and-drying model in simulating the Kennebec–Androscoggin plume and the circulation in Casco Bay. *Ocean Dynamics 60,* 2, 341–357. 21

YALCINER, A. C., IMAMURA, F., AND SYNOLAKIS, C. E. 2011. Amplitude evolution and runup of long waves: Comparison of experimental and numerical data on a 3D complex topography. In *Advanced Numerical Models For Simulating Tsunami Waves And Runup*, P. L. Liu, H. Yeh, and C. Synolakis, Eds. World Scientific Publishing Company, Chapter 9, 243–247.

ZHANG, J., LIN, J., SUN, X., AND CHEN, C. 2009. Application of an improved wetting and drying scheme in POM. In *Proceedings of the 2009 International Conference on Energy and Environment Technology*. Vol. 2. IEEE Computer Society, Washington, DC, 427–432. 21

ZHU, C., BYRD, R. H., LU, P., AND NOCEDAL, J. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software 23,* 4, 550–560. 17