Imperial College London

Department of Computing

Mixed-Integer PDE-Constrained Optimization

Author: Christian Wesselhoeft
Advisor: Dr. Ruth Misener

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science (Machine Learning) of Imperial College London

September 8, 2017

Abstract

Mixed-integer PDE-constrained optimization (MIPDECO) is a flexible framework with a multitude of applications including tidal and wind turbine micro-siting, pharmaceutical business operations and drug production, disaster recovery, and solid product creation, among others. Yet due to the daunting nature of MIPDECO – namely, the combination of integer programming and partial differential equations in a single optimization model – research and application of MIPDECO algorithms is a largely unexplored field. This work provides a palatable introduction to MIPDECO and the mathematical components which constitute it, along with two optimization problems which have been formulated in a MIPDECO framework. Section 1 provides a general introduction to the project, including goals, challenges faced, and contributions, along with a series of tables summarizing all mathematical notation used in the rest of the work. Section 2 presents a mathematical introduction to optimization, partial differential equations, and MIPDECO. Section 3 extends this mathematical background by presenting the Source Inversion problem in a MIPDECO framework, and novel gradient-based algorithms with which to solve it. Section 4 follows with a detailed overview of the hyperparameters of the Source Inversion problem, which is then solved using both branch and bound, a mixed-integer nonlinear programming (MINLP) algorithm, and four gradient-based algorithms. A detailed analysis of time complexity and accuracy for each algorithm is offered. Section 5 introduces tidal turbines and tidal stream turbine optimization – a real-world MIPDECO problem. After reviewing the problem layout from a high level, previous gradient-based optimization methodologies are reproduced, focusing on maximizing power from a field of tidal turbines. A few subsections are dedicated to formulating a new objective function to incorporate time-discounting, revenue, and cost into the model, and a hybrid two-step MIPDECO-based formulation of the optimization problem is presented. Finally, I compare the performance of several gradient-based (MIPDECO and non-MIPDECO formulations) optimization routines using the new profit-based objective functional, ultimately coming to the conclusion that MIPDECO methods for tidal stream turbine optimization are far more efficient, but slightly less accurate than the conventional discrete algorithm, and MIPDECO offers a significant improvement over the former continuous and discrete hybrid model presented by different authors in a previous work. The last section summarizes the results and scope of this project, and offers direction for further research.

Acknowledgements

I would like to extend my sincere thanks to Dr. Ruth Misener for all of her advice and guidance in both the early research-oriented and late presentation-oriented phases of the MSc project. Additionally, I would like to thank Dr. David Ham for his help around the use of partial differential equations and the open source package dolfin-adjoint.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Process optimization is pervasive in human proceedings. A business owner, for instance, wishes to maximize the profit of the business. A construction manager wants to minimize the time it takes to complete a project, subject to certain quality standards. A military commander desires to minimize the financial, human, and ethical cost of a war, while still ensuring that their side obtains victory. In order to solve any of these problems, one must mathematically formulate the problem (which may be difficult – how can one quantize ethics?) by defining decision variables (the inputs), and the objective function (which returns the value one is attempting to maximize or minimize, given the inputs). After a problem is defined and formulated, one can solve the problem by finding peaks (maxima) or troughs (minima) of the objective function using a host of mathematical techniques. While this overarching framework of mathematical optimization is solidly grounded, the unrelenting emergence of newfangled real-world problems necessitates the development of novel optimization methodologies.

One such problem emerges from the field of renewable energy: solar, wind, and tidal power, in particular, hold immense – and untapped – promise for the future of energy generation. As Magagna and Uihlein note, 'despite a high potential associated with ocean energy worldwide, the electricity production from ocean energy is negligible' [1]. Although clean, environmentally-friendly, and renewable energy is seen as socially and politically desirable, heretofore there has been little economic impetus for renewable energy projects, except in geographically favorable areas (i.e. wind or tidal turbines in areas with exceptionally strong winds or tides, such as Canada's Bay of Fundy). The strong headwind to widespread renewables adoption is the relative cost – a 2014 report notes that the levelized cost of energy generation for tidal energy is 3x that of established energy generation methods. Also noted, however, is that the cost of presently-nascent modes of renewable energy generation is projected to fall relatively faster than the costs of traditional generation methods as solar, wind, and tidal technologies are further developed [2]. One arm of cost-saving technological improvement lies in the hardware itself – increasing the efficiency of solar panels, the resiliency of tidal turbines, and so on. Another crucial component is efficiency as a function of location: given a certain set of hardware (turbines, solar panels), how can one best place the components of this set in order to extract the most power, or to derive the most profit? This MSc individual project is constructed with tidal energy in mind, and seeks to find a better technique for maximizing power extraction and lifetime profit of an array of tidal turbines situated in a given domain, subject to the physical constraints of turbines, tides and water.

While solving this problem is the project's telos, we require a significant mathematical foundation before tackling tidal stream turbine optimization (TSTO), which relies heavily on partial differential equations (which model the movement of the tides and tidal wakes of the turbines), gradient-based-optimization (which allows us to efficiently solve a TSTO problem), and mixed-integer programming techniques (which allow us to individually resolve locations of discrete turbines). In light of this, I will present an extensive introduction to gradient-based and integer optimization and partial differential equations in order introduce the primary tool of this project, mixed-integer PDE-constrained optimization (MIPDECO). Next, I will dive into the relatively simple Source Inversion problem in order to demonstrate the power of MIPDECO – and its shortfalls – by benchmarking novel MIPDECO algorithms against each other and against branch and bound, a standard mixed-integer nonlinear programming (MINLP) algorithm; the optimization problems are constructed using open-source software FEniCS, CasADi, and dolfin-adjoint, while existing implementations of gradient-based algorithms from SciPy and IPOPT are used. With the mathematical underpinnings of MIPDECO explicated, I will present a detailed introduction to tidal turbines and optimization thereof, recreating previous work with modified parameters. Finally, I will introduce and solve my own formulation of the tidal stream turbine optimization (TSTO) problem utilizing MIPDECO techniques. Previous and novel TSTO algorithms are implemented in an extended version of the open-source software OpenTidalFarm. Results and analysis will follow, ultimately concluding that MIPDECO provides a useful and flexible framework which sacrifices a small measure of accuracy for massive improvements in time complexity.

1.1 Goals

1. **Simplify the MIPDECO process.** One aim of this project is to break MIPDECO down into a palatable process and provide MIPDECO algorithms which are benchmarked against branch and bound, a well-known mixed-integer nonlinear programming (MINLP) algorithm.

2. **Combine integer and gradient based optimization.** Due to the integer variables inherent to MIPDECO, I want to find a fusion of integer programming and gradient-based optimization which strikes a satisfying balance between accuracy and computational efficiency.

3. **Provide solved problems and prove MIPDECO is useful.** The Source Inversion section seeks to present a detailed solved MIPDECO problem, as accessible examples are few and far between. In order to highlight the applicability of MIPDECO in the real world, this paper seeks to present novel MIPDECO techniques for tidal stream turbine optimization which perform well against the state-of-the-art methods.

1.2 Challenges

1. **Lack of background work.** The primary challenge of this project lies in the lack of any background work in MIPDECO. Though a multitude of real-world optimization problems present themselves naturally in a MIPDECO framework, researchers have tended to recast these problems in a simpler mould before solving them with more familiar techniques.

2. **Integers and gradients.** Many of the problems I solved were computationally intractable in closed form, as a huge number of integer decision variables resulted in a combinatorial explosion for MINLP algorithms such as branch and bound. As a result, I often applied gradient-based optimization to continuous relaxations of integer variables. Consequently, solution methods result in a trade off: gradients and continuous relaxations ensure computational efficiency, but MINLP and integer variables (if tractable) ensure accuracy. Frustratingly, the trade-off is inherently subjective – the desire to maximize accuracy or

computational efficiency is specific to the problem being solved and the researcher's desired accuracy and time constraints.

3. **Poorly posed problems.** Depending on the exact formulation used, the Source Inversion and MIPDECO problems can contain independent variables which scale with the fineness of the discretized domain (the mesh) which define the bounds of the optimization problem. As the mesh fineness increases, these problems become computationally intractable quite quickly.

4. **Computational time/memory complexity.** Even without mesh-dependent independent variables, the size of MIPDECO problems (tens of thousands of independent variables) can make them quite difficult to solve, with a laptop computer either running out of memory, or taking 24+ hours to solve a single optimization problem.

5. **Error tolerance.** Much of chapters 3 and 4 are devoted to comparing gradient-based algorithms to branch and bound. The runtime of gradient-based algorithms is highly dependent on a prespecified error tolerance, however, while branch and bound always finds an exact answer through an intelligent global search. Consequently, comparing algorithm runtimes can be a bit misleading, as there is no objectively 'correct' error tolerance.

1.3 Contributions

1. **Novel algorithms for the Source Inversion problem.** There exist very few examples of formulated and solved MIPDECO problems. Sections 3 and 4 provide a detailed explanation of how to set up and solve a MIPDECO using gradient-based approximation techniques, while benchmarking gradient-based algorithms against branch and bound implemented in CasADi using IPOPT, a high-performance MINLP solver.

2. **Integration of cost modeling into a tidal stream turbine optimization model.** Section 5 introduces a new objective function for tidal stream turbine optimization which attempts to strike a balance between accuracy and simplicity, incorporating the levelized cost of energy generation, inflation, and time-discounting. Unlike previous propositions, the objective function used in this work is not at the behest of a user-supplied profit margin, and is not burdened by the complexity of using a cable-routing problem driven by a genetic algorithm to determine cost.

3. **Providing a novel MIPDECO framework for tidal stream turbine optimization in OpenTidalFarm.** Building off the work of Funke et al., I formulate a modified version of the tidal stream turbine problem as MIPDECO, and introduce a novel algorithm which retains most of the efficiency of Funke et al.'s continuous model, and much of the accuracy of their discrete model. For this purpose, I added support for MIPDECO problems to OpenTidalFarm, an open source software for tidal stream turbine optimization.

1.4 Mathematical Notation and Abbreviations

Before a mathematical introduction to optimization techniques and partial differential equations, the following reference tables outline the notation that will be used in this project. Note that the Source Inversion and TSTO problems have their own tables for problem-specific notation, and that the final table contains the definitions of the (many) abbreviations which appear often in this project.

Basic Notation	**Definition**
$\boldsymbol{x}$	A bold lower-case letter denotes a vector which exists in the appropriate vector space.
$\boldsymbol{x}^T$	The transpose of the vector $\boldsymbol{x}$.
x_i^t	The subscript on a lower-case letter always denotes a particular component of a vector, while a superscript denotes its position in time. Thus x_i^t is the i'th component of $\boldsymbol{x}$ at time t.
$\boldsymbol{A}$	An upper-case letter denotes a matrix which exists in the appropriate vector space. Similarly to vectors, a 'T' superscript denotes the transpose. while a 't' supercript denotes time. A_{ij}^t represents the ij'th component at time t.
$\mathbf{0}$	A bold zero denotes the zero vector which exists in the appropriate vector space.
$\boldsymbol{g}(\boldsymbol{x})$	A boldface function is used when the inputs are mapped to a vector. In other words: $\boldsymbol{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where n and m are positive integers greater than 1.
$\mathcal{C}^k$	The space of k-times continuously differentiable functions
$\mathcal{O}(n)$	The asymptotic memory or time complexity of an algorithm as a function of the algorithm's n inputs.
$\langle .,. \rangle_\Omega$	The inner product over vector space Ω.
$\lvert\lvert . \rvert\rvert_2^2$	The L-2 norm squared.
$\neg\Box(x = y)$	x is not required to equal y

PDE Notation	**Definition**
$\boldsymbol{u}$	A function of a vector of decision variables and the partial derivatives of that function with respect to the decision variables. In long form, this may be written as $\boldsymbol{u}(x, \boldsymbol{u}_x, \boldsymbol{u}_{xx}, ...)$ where $\boldsymbol{u}_x$ denotes the gradient of $\boldsymbol{u}$ with respect to $\boldsymbol{x}$.
$\mathcal{H}$	A possibly infinitely dimensional Hilbert Space. Subscripts are used to identify unique Hilbert Spaces when more than one is involved.
$\mathcal{V}$	The finite Hilbert Space derived from $\mathcal{H}$ using the finite element method.
$\mathcal{F}(\boldsymbol{u}, \boldsymbol{x})$	A system of partial differential equations which can be parametrized by its decision variables $\boldsymbol{x}$, and the function $\boldsymbol{u}$ and its partial derivatives with respect to $\boldsymbol{x}$.
$\mathbb{F}(\boldsymbol{u}, \boldsymbol{x})$	The discrete system of partial differential equations (i.e. the PDE after the finite element method has been applied).
Ω	The domain over which a system of partial differential equations holds.
$\delta\Omega$	The boundary of the domain Ω.
Θ	The discretized version of domain Ω. In other words, the result of applying the finite element method to Ω.

Optimization Notation	**Definition**
$\arg\min_{\boldsymbol{x}} \ f(\boldsymbol{x})$	A minimization problem posed as such: find $\boldsymbol{x}$ such that the objective function f is minimized.
$\mathcal{J}(\boldsymbol{u}, \boldsymbol{w})$	The objective functional in a PDE-constrained optimization problem or Mixed-Integer PDE-constrained optimization (MIPDECO) problem.
$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$	The objective Lagrangian function with decision variables $\boldsymbol{x}$ and Lagrangian multipliers $\boldsymbol{\lambda}$.
$\boldsymbol{x}^*$	The solution vector to an optimization problem. In other words $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$.
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$	The gradient, or vector of first-order partial derivatives of function f with respect to decision variable vector $\boldsymbol{x}$.
$\nabla_{\boldsymbol{xx}} f(\boldsymbol{x})$	The Hessian, or matrix of second-order partial derivatives of function f with respect to decision variable vector $\boldsymbol{x}$.
$\mathbb{H}$	The approximation of the Hessian matrix as used in the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization method.
$\Delta f(\boldsymbol{x})$	The Laplacian, or sum of second-order partial derivatives of function f with respect to decision variable vector $\boldsymbol{x}$.
$\int_\Omega$	The integral over the domain Ω.

Source Inversion Notation	**Definition**
Ω	The continuous domain (a unit square).
Θ	The discrete domain, (a unit square), represented by a m x m mesh, where m is a positive integer.
n	Source function number parameter. An evenly spaced subsquare of n x n source functions is embedded into the mesh.
m	Mesh vertex parameter; mesh Θ has $m+1$ x $m+1$ vertices.
$\boldsymbol{w}$	The vector of binary decision variables corresponding to the centroids of source functions.
$\bar{\boldsymbol{u}}$	The continuous reference PDE. After applying the finite element method, this changes to $\bar{\boldsymbol{u}}^d$, the discrete reference PDE.
$\boldsymbol{u}$	The continuous source PDE. After applying the finite element method, this changes to $\boldsymbol{u}^d$, the discrete source PDE.
a	The scaling hyperparameter for Gaussian source functions.
$\mathbb{V}$	The set of all vertices in the domain, Ω.
σ	The variance hyperparameter for Gaussian source functions.
α^*	The penalty parameter for the penalty objective function.
S	The maximum number of activated source functions.
$\boldsymbol{A}$	The constraint matrix for the cont. algorithm (sum control).
ϵ_{L1}	The error metric which measures the difference between the solution to the reference PDE and the solution to the reconstructed reference PDE.

Tidal Turbine Notation	**Definition**
C_T	Thrust coefficient for a single turbine.
A_T	Diameter of the area swept out by the blades of a turbine.
t	$t \in (0, T)$, the simulation period.
Ω	The turbine farm state space.
ρ	fluid (water) density.
$\boldsymbol{u}$	Velocity of the flow.
ν	Free surface displacement.
g	Acceleration due to gravity.
c_b	Constant background bottom friction.
c_t	Enhanced friction of parametrized turbines.
H	Resting water depth.
$\boldsymbol{m}$	A vector representing the location of discrete turbines on a 2-dimensional grid - i.e. $[x_1, y_2, x_2, y_2, ...x_n, y_n]$.
$\boldsymbol{w}$	A vector of binary variables corresponding to vertices of the discretized farm area. If $w_i = 1$, a turbine is present at vertex i; else, no turbine is present.
$P(.)$	Power or profit as a function of discrete turbine location or binary variables.
κ	Denotes time stationarity; κ is 0 if stationary, else 1.
$LCOE$	The levelized cost of energy (MWh).
O_t	Energy output (MWh) at time t.
C_t	Raw cost of energy (MWh) produced at time t.
$E_t(\boldsymbol{u}, .)$	Energy produced at time t (MWh) as a function of the velocity of flow $\boldsymbol{u}$ and location variables $\boldsymbol{m}$ or binary location variables $\boldsymbol{w}$.

Abbreviations	**Definition**
PDE	Partial Differential Equation
MIPDECO	Mixed-Integer PDE-Constrained Optimization
MIP	Mixed Integer Programming
MINLP	Mixed Integer Nonlinear Programming
BFGS	Broyden-Fletcher-Goldfarb-Shanno Algorithm
SQP/SLSQP	Sequential (Least Squares) Quadratic Programming
FEM	Finite Element Method
FDM	Finite Difference Method
FVM	Finite Volume Method
DTP	Dynamic Tidal Power
ADM	Actuator Disc Momentum Theory
TSTO	Tidal Stream Turbine Optimization

Chapter 2

Mathematical Background

2.1 An Introduction to Optimization

Before diving into MIPDECO, it is necessary to introduce the basics of constrained optimization formulations and algorithms which are used to solve such problems. I will assume the reader has knowledge of linear algebra and matrix calculus. For a more in-depth examination of linear algebra, matrix calculus, and optimization, the interested reader may also refer to Bretscher [3] for a linear algebra refresher, Peterson & Peterson [4] for a review of matrix calculus, and to Chong & Zak [5] as a source on mathematical optimization and convexity.

2.1.1 Constrained and Mixed-Integer Optimization

A continuous optimization problem may be formulated as a maximization or minimization problem subject to constraints. It is important to note that maximization and minimization problems can be interchanged to find the same solution, as the equality $\max\limits_{x} f(x) = -\min\limits_{x} \; -f(x)$ always holds. To construct an optimization problem, we can take decision variable(s) $\boldsymbol{x} \in \mathbb{R}^n$ as the inputs. Given an objective function $f : \mathbb{R}^n \to \mathbb{R}$, and constraints $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{h}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^p$, a constrained optimization problem may be written as:

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \quad s.t. \quad \begin{cases} \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} \end{cases} \tag{2.1}$$

In some situations, the values of some or all of the decision variables may be constrained to be integers. For instance, if decision variables $\boldsymbol{x} \in \mathbb{R}^3$ represents the quantity of Nitrogen, Potassium, and Phospates, which are purchased in kilograms and mixed together in a certain ratio in order to create fertilizer, a farmer may wish to maximize the total amount of fertilizer created subject to a certain budget/cost of each ingredient and the fact that each ingredient must be purchased in a non-negative integer amount. This integer-constrained optimization problem (and any general problem) is identical to the continuous optimization problem in Equation 2.1, with the stipulation that some (mixed integer programming or MIP) or all (pure integer programming or IP) of the decision variables must be integers. Formally, if given $\boldsymbol{x} \in \mathbb{R}^{n+k}$, function $f : \mathbb{R}^{n+k} \to \mathbb{R}$, and constraints $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^{n+k} \to \mathbb{R}^m$, $\boldsymbol{h}(\boldsymbol{x}) : \mathbb{R}^{n+k} \to \mathbb{R}^p$, the optimization problem takes the familiar form, with k integer variables (and $n = 0$ continuous variables) for an IP, and k integer variables and n continuous variables in a MIP. We may write the optimization problem as (assuming that $k + n = N$):

$$\max_{\boldsymbol{x}} \; f(\boldsymbol{x}) \quad s.t. \quad \begin{cases} \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} \\ x_i \in \mathbb{Z} \;\; \forall i \in 1, ..., K \\ x_j \in \mathbb{R} \;\; \forall j \in K+1, ..., N \end{cases} \tag{2.2}$$

Now let us take the example of the farmer who is trying to make as much *quality-weighted* fertilizer as possible subject to his continuous budgetary and integer quantity requirements. If we assume that the quality of fertilizer varies depending on the location at which it is stored at the farm (i.e. the weather conditions the stored fertilizer is subject to) and the duration for which it is stored and unused, we can now formulate this problem as a mixed-integer PDE-constrained optimization problem. In other words, the quality-adjusted quantity of fertilizer created now depends on decision variables (some of which are integer-constrained) which also vary over time and space (or, in other words, a system of partial differential equations which exist in a domain encapsulated by the farm). If we generalize this idea, we may formalize a mixed-integer PDE-constrained optimization (MIPDECO) problem as an extension of Equation 2.2 (a detailed treatment of PDEs follows this section). In mathematical terms, it is important to note that the decision variable $\boldsymbol{x}$ now exists in a possibly infinite-dimensional Hilbert space (potentially rendering direct optimization impossible), as it is affected by the PDE in a varying manner over the entire domain. The resulting optimization problem takes the form (where $\mathcal{F}$ denotes the PDE system, and K and N are potentially infinite):

$$\min_{\boldsymbol{u},\boldsymbol{x}} \; J(\boldsymbol{u}, \boldsymbol{x}) \quad s.t. \quad \begin{cases} \mathcal{F}(\boldsymbol{u}, \boldsymbol{x}) = 0 \\ \boldsymbol{g}(\boldsymbol{x}) = 0 \\ \boldsymbol{h}(\boldsymbol{x}) \leq 0 \\ x_i \in \mathbb{Z} \;\; \forall i \in 1, ..., K \\ x_j \in \mathbb{R} \;\; \forall j \in K+1, ..., N \\ \boldsymbol{x}, \boldsymbol{u} \in \mathcal{H}_1, \mathcal{H}_2 \end{cases} \tag{2.3}$$

Solving this optimization problem in a well-posed situation can be extremely difficult, and to cause further headache, MIPDECO problems are often nonconvex and nonlinear! The next few sections will present a high level analysis of solution techniques, before delving into the algorithmic and mathematical specifics applied to particular problems in the following chapters.

2.1.2 Solving a Continuous Optimization Problem

A continuous optimization problem in its most basic form, that is, Equation 2.1 without equality or inequality constraints (i.e. no $\boldsymbol{g}(\boldsymbol{x})$ or $\boldsymbol{f}(\boldsymbol{x})$) can sometimes be solved trivially in closed form by taking partial derivatives with respect to the decision variables and setting the resulting equations equal to zero. In the oft-occurring situation in which a closed form solution is not available, however, an unconstrained optimization problem may be solved using methods derived from three major categories.

1. **Gradient-based, or first-order methods**, are optimization algorithms which only take into account the gradient of the objective function with respect to the decision variables.

2. **Second order optimization algorithms** take the gradient of the objective function *and* the Hessian matrix (the matrix of second-order partial derivatives of the objective function with respect to the decision variables) into account.

3. **Derivative-free methods** do not take any derivative information into account, relying solely on heuristics, global search techniques, and/or privileged information such as the Lipschitz constant or convexity when optimizing.

It is important to note that each optimization class and the algorithms therein have special requirements for convergence, an examination of which is out of the scope of this project, though the interested reader can refer to Chong & Zak's Optimization textbook [5]. The following table presents some basic requirements and properties for each class of optimization algorithms. Note that there is often a trade-off in computational complexity: as a general guideline, second-order algorithms require the fewest number of iterations to converge (i.e. they have the lowest time complexity), followed by gradient methods and then derivative free methods. On the other hand, second-order methods require the most memory (i.e. highest memory complexity) as they need to store gradient and hessian information, followed by gradient and then derivative-free methods. As I will show shortly, in the case of MIPDECO problems, gradient-based methods provide the optimal trade-off in terms of minimizing time and memory complexity.

Table 2.1: Optimization Algorithm Class Requirements and Properties

First-Order	Second-Order	Derivative-Free
Objective function $\in \mathcal{C}^1$	Objective function $\in \mathcal{C}^2$	O.F. differentiability optional
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ (gradient)	$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, $\nabla_{\boldsymbol{xx}} f(\boldsymbol{x})$ (grad. and Hessian)	N/A
Moderate Memory, Time	High Memory, Low Time	Low Memory, High Time

While the above optimization methods can be applied directly to unconstrained optimization problems, introducing equality constraints necessitates the usage of an augmented objective function known as the Lagrangian, which incorporates the offending constraints into the objective function as a penalty term (similar to a regularization term). Taking Equation 2.1 as an example (and disregarding the inequality constraint), the optimization problem is transformed (with the vector $\boldsymbol{\lambda}$ being composed of Lagrangian multipliers) to:

$$\min_{\boldsymbol{x}} \ \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) \tag{2.4}$$

If a regularity condition and the Lagrange condition holds, this problem may be solved in closed form. If the necessary conditions do not hold, Equation 2.4 is treated as an unconstrained optimization problem and can be solved using one of the optimization method classes in Table 2.1.

The inclusion of inequality constraints (with or without equality constraints) results in a similar optimization problem as in Equation 2.4 (note that there are separate Lagrangian multipliers for inequality and equality constraints, as the corresponding optimality conditions are different):

$$\min_{\boldsymbol{x}} \ \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{h}(\boldsymbol{x}) \tag{2.5}$$

The presence of inequality constraints necessitates the deployment of the Karush-Kuhn-Tucker conditions to solve the problem in closed form if it is nicely posed (i.e. a host of regularity conditions are satisfied) [7] [8]. Unfortunately, fulfillment of these conditions is generally not the case for my work, and thus the application of optimization methods derived from Table 2.1 is required.

2.1.3 Solving an Integer Optimization Problem

The inclusion of integer decision variables greatly complicates any optimization problem, as all derivative-based algorithms are inapplicable due to the lack of continuity and differentiability of the objective function with respect to the integer variables. Consequently, there exists a limited range of options. The first, and easiest, method, is to first solve a continuous relaxation of the optimization problem using gradient-based or second order methods, followed by a rounding or otherwise heuristic to transform the relevant optimal continuous decision variables into integer variables. There are no guarantees, however, that rounding each decision variable (or using a different conversion heuristic) of the optimal continuous solution to the nearest integer will result in the optimal integer solution.

In order to obtain a more precise solution without first solving the continuous relaxation, one can add a penalty term to the objective functional which increases (min) or decreases (max) the objective function when the integrality constraints are violated. For instance, let us consider decision variables $\boldsymbol{x}$ where $x_i \in \{0, 1\}\ \forall i$, and a minimization problem such as in 2.1 but without any constraints (though this method easily extends to handle constraints). The objective functional may be modified with a penalty term as such:

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) + \sum_{i=1}^{N} \alpha_i^* x_i(1 - x_i) \tag{2.6}$$

As we are trying to minimize the objective function, for each component of $\boldsymbol{x}$ that is not precisely 0 or 1, a penalty will be added to the objective function, thus discouraging any non-binary values of the decision variables. A drawback to this method, however, is finding the precise α_i^* which does not under-penalize x_i for violating the binary constraints, but also does not penalize violation to the extent that the optimization problem only finds feasible configurations of $\boldsymbol{x}$. As the complexity of finding α_i^* increases in the number of decision variables, a simpler – and less powerful – method is to just apply a single value α to each x_i instead of custom-fitting each term. If we do find the optimal value(s) of α_i or α, then any gradient or second-order optimization algorithm can be used to solve for integer or approximate-integer solutions.

The last paradigm for solving optimization problems with integer variables is based around using derivative-free algorithms that specialize in finding integer solutions, such as branch and bound or branch and cut. These mixed-integer (nonlinear) programming (MIP/MINLP) algorithms use divide and conquer methods (branch and cut adds cutting planes) to exhaustively search the domain of an optimization problem for *exact* and optimal integer solutions. As a principal impetus of this project is to attempt to solve MIPDECO problems efficiently with a gradient-based approach, as compared to the accurate (and often computationally prohibitive) MINLP algorithms, I will devote little time to an analysis of the methods used, aside from an overview of branch and bound, the MINLP baseline which my gradient-based methods will compete against. For an overview of the history of integer programming and detailed descriptions of the algorithms used, see Doig [10], Gomory [9], Dakin [11], Cornuéjols [12], and Padberg and Rinaldi [13].

2.1.4 Optimization Algorithms and Complexity

Solving an optimization problem requires us to find the configuration of decision variables $\boldsymbol{x}^*$ which results in the lowest value (if a minimization problem) or highest value (if a maximization problem) of the objective function. When solving the problem in closed form is not an option, one instead chooses an initial configuration of the decision variables $\boldsymbol{x}^{init}$. If using a derivative-free method, then some heuristic is required to guide exploration from $\boldsymbol{x}^{init} \rightarrow \boldsymbol{x}^1 \rightarrow \ldots \rightarrow \boldsymbol{x}^*$. In principal, the simplest option would be to evaluate every possible configuration of decision variables in the domain, storing each configuration and the corresponding objective value function; we would then simply pick best configuration. In MIPDECO problems, however, the domain is so large that this technique – a 'global search' – is infeasible. For instance, in the least computationally demanding problem I will be solving, $\boldsymbol{x} \in \mathbb{R}^{16}$ with $x_i \in \{0, 1\}$. Even in this simple scenario, a global search would require 2^{16} functional evaluations – clearly, this approach is not scalable.

Having crossed out the derivative-free class, the two remaining methods both search the domain to find $\boldsymbol{x}^*$ using gradient and/or second-order information to guide the search direction. Importantly, it can be shown that the gradient of a function points in its direction of maximal increase (and thus the negative gradient points in the direction of the maximal decrease) [5]. Now consider a generic unconstrained minimization problem with initial decision variable configuration $\boldsymbol{x}^{init}$. A gradient-based optimization algorithm will derive a descent direction based on the gradient of objective function with respect to the current configuration of the decision variables (starting with $\boldsymbol{x}^{init}$), and then 'step' in that direction with step-size α_t, which is chosen during each iteration of the descent algorithm by an exact line search or a heuristic approximation thereof (such as backtracking) until some termination condition is reached (generally an error tolerance for the change in objective function value, gradient value, or decision variable value between iterations, i.e. $||\boldsymbol{x}^t - \boldsymbol{x}^{t-1}|| \leq \epsilon$). A second-order optimization algorithm follows same process, except that it considers gradient *and* second-order information when deriving the descent direction. Formally, if we are given decision variables $\boldsymbol{x} \in \mathbb{R}^n$ and during each iteration choose a descent direction $\boldsymbol{d}^t \in \mathbb{R}^n$, a general descent algorithm is:

Algorithm 1 Descent Algorithm (Minimization)

```
1: Begin with objective function $\boldsymbol{f}(\boldsymbol{x})$, $\boldsymbol{x}^{init}$, and set time $t = 0$
2: while termination condition not satisfied do
3:     Derive descent direction $\boldsymbol{d}^t$
4:     Choose $\alpha^t$ by a suitable line search
5:     $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \alpha^t \boldsymbol{d}^t$
6:     $t++$
7: return $\boldsymbol{x}^{t+1}$
```

The choice of $\boldsymbol{d}$ depends on the algorithm used, and can incorporate gradient information (steepest descent, conjugate gradient descent, etc.) and second order information, or approximations thereof. It is important to note the varying space complexity of different descent directions choices, resulting in the inapplicability of second-order methods to the PDE-constrained optimization problems I will be considering.

To illustrate the space complexity issue, we can consider Newton Descent, which chooses the descent direction $\boldsymbol{d}^t = -[\nabla_{\boldsymbol{xx}} \boldsymbol{f}(\boldsymbol{x}^t)]^{-1} \cdot \nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}^t)$ during each iteration. The construction of $\boldsymbol{d}^t$ requires the inversion of the Hessian matrix – an $\mathcal{O}(n^3)$ task in n decision variables. For large optimization problems (such as those with PDE constraints) which may include thousands or hundreds of thousands of decision variables, any sort of iterative second order optimization method is computationally intractable. While the Cholesky Decomposition or Woodbury Formula can be used to calculate the inverse directly and more efficiently the baseline memory requirements (storing the Hessian at each iteration is $\mathcal{O}(n^2)$) are ultimately too high to use second order optimization methods reliably. With a host of mathematical tricks, however, first order gradient-based methods which require calculation of $\boldsymbol{d}^t = \nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}^t)$ – only an n x 1 gradient with n decision variables – or second order *approximation* methods using only gradient information (such as the Banno-Farb-Goldstein-Shanno (BFGS) and quasi-Newton algorithms) do remain feasible. If one recalls the trade-off described earlier in this section, we may think of second-order methods of being extremely time-efficient, but requiring a huge amount of memory to use, while the written-off global methods are extremely space-efficient, but require a large amount of time (many iterations) to use. Gradient-based optimization algorithms, however, are moderately time-and-space efficient. The interested reader may refer to Chong and Zak's *Introduction to Optimization* [5] for extensive analysis of optimization methods and their time complexity.

Table 2.2: Optimization algorithms with $\mathcal{O}(n)$ space complexity in n variables

$\mathcal{O}(n)$
Gradient Descent
Stochastic/Conjugate Gradient Descent
Sequential Quadratic Programming
Secant Method
BFGS/L-BFGS-B

As a result of the discussion above, only optimization algorithms whose *space* complexity is $\mathcal{O}(n)$ are feasible. Despite the oft-lower time complexity of Newton Descent and similar methods, the memory requirements of $\mathcal{O}(n^3)$ and $> \mathcal{O}(n^3)$ algorithms are too large to be able to process PDEs; as noted before, derivative-free methods have prohibitively high time complexity.

An additional importance to note is that MIPDECO problems are inherently nonconvex due to the integer variables – though their continuous relaxations may be convex (see the Appendix for a detailed explanation of convexity and global/local extrema) – and thus do not have a single unique global minima. Without the aforementioned global search, then, gradient-based algorithms will not necessarily return the globally optimal $\boldsymbol{x}^*$, but rather a locally optimal $\boldsymbol{x}^*_{local}$. In an attempt to find the best possible local minima, we can vary the initial parameters of the optimization problem. That is, if we run a gradient-based (or otherwise) optimization algorithm on an optimization model using a different $\boldsymbol{x}^{init}$ each time, then the output of the algorithm $\boldsymbol{x}^*$ may terminate at a different local optimum during each run of the optimization algorithm. If we iterate, say, 10 times, with random (or intelligently-informed) initialization values of the decision variables, we may simply pick the configuration of the decision variables in the resulting set which yield the best objective function value (i.e. those decision variables corresponding to the best local minima). While one may note that there is a significant amount of extra computational effort in this strategy, the memory complexity is maintained–that is, a $\mathcal{O}(n)$ gradient-based algorithm is still $\mathcal{O}(n)$, while the time complexity increases *linearly* in the number of algorithm re-runs. For instance, if a gradient based algorithm has time complexity $\mathcal{O}(n)$, then running m iterations results in the entire pipeline having complexity $m \cdot \mathcal{O}(n)$. Any algorithm will then asymptotically retain its original time and memory complexity, supporting the original justification for using only algorithms with $\mathcal{O}(n)$ memory complexity.

2.1.5 Relevant Optimization Algorithms

In the following MIPDECO case studies presented in chapters 3, 4, or 5, I will use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) and sequential quadratic programming (SQP) algorithms for unconstrained and constrained optimization, respectively, as implemented in Python's SciPy package. It is important to note that these are both gradient-based nonlinear optimization algorithms which use first-order gradient information to approximate the Hessian matrix, a crucial component of Newton descent. In other words, the algorithms I am using are gradient-based approximations to Newton descent (BFGS) and Newton descent with equality and inequality constraints (SQP). The performance of optimization routines incorporating these algorithms will be compared to the MINLP algorithm branch and bound, which I have implemented in Python using the open-source CasADi package with IPOPT as the MINLP optimization module therein.

2.1.6 Newton Descent

Consider the nonlinear unconstrained optimization problem with $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and decision variable $\boldsymbol{x} \in \mathbb{R}^n$. For now, we do not consider constraints.

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \tag{2.7}$$

Given an initial choice of decision variables $\boldsymbol{x}^0$, Newton Descent uses the descent direction $\boldsymbol{d}^t = -[\nabla_{\boldsymbol{xx}} \boldsymbol{f}(\boldsymbol{x}^t)]^{-1} \cdot \nabla_x f(\boldsymbol{x}^t)$. The Newton descent algorithm then proceeds as:

Algorithm 2 Newton Descent (Minimization)

```
1: Begin with objective function $\boldsymbol{f}(\boldsymbol{x})$, initial guess for decision parameters $\boldsymbol{x}^0$, time $t = 0$
2: while termination condition not satisfied do
3:     Derive descent direction $\boldsymbol{d}^{\boldsymbol{t}} = -[\nabla_{\boldsymbol{xx}} \boldsymbol{f}(\boldsymbol{x}^t)]^{-1} \cdot \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$
4:     Find optimal step size $\alpha^t$ through a line search
5:     $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \alpha^t \boldsymbol{d}^t$
6:     $t++$
7: return $\boldsymbol{x}^{t+1}$
```

Newton Descent converges in quadratic time, though may not converge to an optimal solution if the initial guess $\boldsymbol{x}^0$ is too far from the optimal configuration of decision variables – this will be a crucial point when applied to finding optimal integer decision variables in a MIPDECO. While the above method is ideal in terms of accuracy (despite lack of convergence guarantees), the calculation of the descent direction involves the inversion of the Hessian matrix, which is $\mathcal{O}(n^3)$ in n decision variables. Consequently, we turn to other algorithms, such as BFGS, to use gradient information to approximate $[\nabla_{\boldsymbol{xx}} \boldsymbol{f}(\boldsymbol{x}^t)]^{-1}$.

2.1.7 Broyden-Fletcher-Goldfarb-Shanno Algorithm (BFGS)

Taking the optimization problem defined in the previous section, the BFGS method defines the variables $\boldsymbol{s} = \boldsymbol{x}^{t+1} - \boldsymbol{x}^t$ and $\boldsymbol{y} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{t+1}) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$, and generally initializes the estimated Hessian matrix $\mathbb{H}^0$ to a n x n identity matrix where the decision variables $\boldsymbol{x} \in \mathbb{R}^n$; alternatively, the estimated Hessian can be initialized as any n x n positive definite symmetric matrix. At each following iteration, the BFGS method approximates the Hessian matrix as:

$$\nabla_{\boldsymbol{xx}} f(\boldsymbol{x}^{t+1}) \sim \mathbb{H}^{t+1} = \mathbb{H}^t + \frac{\boldsymbol{y}\boldsymbol{y}^T}{\boldsymbol{y}^T\boldsymbol{s}} - \frac{\mathbb{H}^t \boldsymbol{s}\boldsymbol{s}^T \mathbb{H}^t}{\boldsymbol{s}^T \mathbb{H}^t \boldsymbol{s}} \tag{2.8}$$

The BFGS algorithm then modifies the aforementioned Newton descent algorithm:

Algorithm 3 BFGS (Minimization)

```
1: Begin with objective function $\boldsymbol{f}(\boldsymbol{x})$, initial guess for decision parameters $\boldsymbol{x}^0$, time $t = 0$
2: Set $\mathbb{H}^0 = \mathbb{I}(n)$ or any other symmetric positive definite matrix
3: while termination condition not satisfied do
4:     Derive descent direction $\boldsymbol{d}^{\boldsymbol{t}} = -[\mathbb{H}^t]^{-1} \cdot \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$
5:     Find optimal step size $\alpha^t$ through a line search
6:     $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \alpha^t \boldsymbol{d}^t$
7:     Compute $[\mathbb{H}^{t+1}]^{-1}$
8:     $t++$
9: return $\boldsymbol{x}^{t+1}$
```

Variants of BFGS include BFGS-B, which allows for the decision variables to take bounds, and L-BFGS-B, which allows bounded decision variables and has a limited memory property so that the approximated Hessian matrix is not actually stored (which can be rather large as it is $\mathcal{O}(n^2)$ in n decision variables), but rather the most k recent $\boldsymbol{s}$ and $\boldsymbol{y}$ vectors, where k is user-defined. While BFGS and its variant are known as well-performing approximations of Newton's method, they can only be applied to *unconstrained* optimization.

2.1.8 Sequential Quadratic Programming (SQP)

Sequential Quadratic Programming (SQP) can be viewed as the natural extension of BFGS to nonlinear constrained optimization problems. Consider the nonlinear constrained optimization problem with $f : \mathbb{R}^n \to \mathbb{R}$, decision variable $\boldsymbol{x} \in \mathbb{R}^n$, equality constraints $\boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^I$ and $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^J$.

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \quad s.t. \quad \begin{cases} \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} \end{cases} \tag{2.9}$$

To solve this problem, the Lagrangian is formed, with multipliers $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) \tag{2.10}$$

At each iteration of SQP, the algorithm solves a quadratic sub-problem to approximate the above Lagrangian. Note that (with notational abuse) the Hessian at time t $\nabla\nabla(\mathcal{L}(\boldsymbol{x}^t, \boldsymbol{\mu}^t, \boldsymbol{\lambda}^t))$ is the matrix of second partial derivatives of the objective Lagrangian with respect to the decision variables *and* both of the multiplier vectors. This is approximated by $\mathbb{H}(\mathcal{L}(\boldsymbol{x}^t, \boldsymbol{\mu}^t, \boldsymbol{\lambda}^t))$ using the BFGS method. Thus at iteration t SQP finds the solution to:

$$\min_{\boldsymbol{d}} \; \boldsymbol{d}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t) + \frac{1}{2} \boldsymbol{d}^T \mathbb{H}(\mathcal{L}(\boldsymbol{x}^t, \boldsymbol{\mu}^t, \boldsymbol{\lambda}^t)) \boldsymbol{d} \quad s.t. \quad \begin{cases} \nabla_{\boldsymbol{x}} g_i(\boldsymbol{x}^t)^T \boldsymbol{d} + g_i(\boldsymbol{x}^t) = \boldsymbol{0} \;\; \forall i \in I \\ \nabla_{\boldsymbol{x}} h_j(\boldsymbol{x}^t)^T \boldsymbol{d} + h_j(\boldsymbol{x}^t) \leq \boldsymbol{0} \;\; \forall j \in J \end{cases} \tag{2.11}$$

The solution to this sub-problem yields the descent direction $\boldsymbol{d}_{\boldsymbol{x}}$ for the decision variables as well as directions $\boldsymbol{d}_{\mu}$ and $\boldsymbol{d}_{\lambda}$) for the equality and inequality constraint multipliers, respectively. Consequently, the multipliers and decision variables are updated and the process repeats until an error tolerance is reached.

Algorithm 4 Sequential Quadratic Programming (Minimization)

```
1: Begin with objective function $\boldsymbol{f}(\boldsymbol{x})$, initial guess for decision parameters $\boldsymbol{x}^t$, time $t = 0$
2: Set $\mathbb{H}^0 = \mathbb{I}(dim(x))$ or any other symmetric positive definite matrix
3: while termination condition not satisfied do
4:     Solve the quadratic subproblem 2.11 for $\boldsymbol{d}_{\boldsymbol{x}}$, $\boldsymbol{d}_{\boldsymbol{\mu}}$, and $\boldsymbol{d}_{\boldsymbol{\lambda}}$
5:     $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \boldsymbol{d}_{\boldsymbol{x}}$
6:     $\boldsymbol{\mu}^{t+1} = \boldsymbol{d}_{\mu}$
7:     $\boldsymbol{\lambda}^{t+1} = \boldsymbol{d}_{\lambda}$
8:     $t++$
9: return $\boldsymbol{x}^{t+1}$
```

Among other favorable qualities, SQP allows for the initial conditions to be infeasible; thus $\boldsymbol{x}^0$ is not required to satisfy the constraint, which is quite useful in the following chapter when attempting to solve the Source Inversion problem.

2.1.9 Branch and Bound

While BFGS and SQP will be used in novel optimization routines in the hopes of dramatically improving solution efficiency with little expense in regards to solution accuracy, these two gradient based algorithms will be compared against branch and bound, a classic divide-and-conquer integer programming technique. We first consider the Mixed-Integer Nonlinear Programming (MINLP) problem:

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \quad s.t. \quad \begin{cases} \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} \\ x_i \in \mathbb{Z} \quad \forall i \in 1...K \\ x_j \in \mathbb{R} \quad \forall j \in K+1...N \end{cases} \tag{2.12}$$

This problem is reformulated as the root problem, in which a continuous relaxation of all integer variables is performed:

$$\min_{\boldsymbol{x}} \; MINLP(root) = f(\boldsymbol{x}) \quad s.t. \quad \begin{cases} \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{x}) \leq \boldsymbol{0} \\ x_i \in \mathbb{R} \quad \forall i \in 1...K \\ x_j \in \mathbb{R} \quad \forall j \in K+1...N \end{cases} \tag{2.13}$$

We denote this to be the root problem, and it corresponds to $MINLP(-\infty, \infty)$, meaning that the value of each integer variable is bounded by $MINLP(-\infty, \infty)$. The branch and bound algorithm then proceeds as follows: after solving the root problem, if the solution is not integral, then two sub-problems are created by branching on one of the variables that is required to be integral, but currently is not. That is, we select x_i^t which currently has constraints $b^- < x_i^t < b^+$ (initially, $b^- = -\infty, b^+ = \infty$) and create new bounds (l^-, u^-) and (l^+, u^+), where $u^- = \lfloor x_i^t \rfloor$ and $l^+ = \lceil x_i^t \rceil$. Two new problems formulated like Equation 2.13 are added to the heap of unsolved problems, with bounds initialized identically to $MINLP(root)$ except that in one problem, $b^- \leq x_i^{t+1} \leq \lfloor x_i^t \rfloor$, and in the other $\lceil x_i^t \rceil \leq x_i^{t+1} \leq b^+$. The process the repeats, with each solution adding two new problems to the heap, unless one of three conditions is satisfied. If the solution is infeasible, then traversal down the branch is immediately stopped. If the solution is an integer solution, traversal down the current branch ceases and the integer solution is compared to the current best integer solution. If the solution is better than the current best solution f_{best}, then f_{best} is updated. Otherwise, if the solution is not integer, but is worse than the best integer solution so far, traversal down that branch also stops, as any integer solution that could be yielded therefrom is guaranteed to be worse than the current integer solution.

One must note that branch and bound is a clever way of performing an exhaustive search fairly efficiently, but still requires solving a massive number of sub-problems. Consequently, while it is guaranteed to return a globally optimal objective function value and decision variable configuration, the algorithm may be computationally infeasible. Naturally, as the number of integer variables increases, so does the number of sub-problems increase, as it is likely that many branch steps will be required for each integer variable x_i. While branch and bound is the foundation for more state-of-the-art techniques (such as branch and cut), I will not be investigating the subtleties and comparison of MINLP algorithms in this project. For a comprehensive literature review on MINLP techniques, consult Belotti et al. [22].

Algorithm 5 MINLP Branch and Bound

```
1: Begin with $f_{opt} = \infty$, error tolerance $\epsilon$, and empty heap $\mathcal{H} = \emptyset$
2: Add $MINLP(-\infty, \infty)$ to the heap
3: while $\mathcal{H}$ is not empty do
4:     Remove problem $MINLP(l, u)$ from the heap
5:     Solve $MINLP(l, u)$ for $\boldsymbol{x}^*$
6:     if $\boldsymbol{x}^*$ is infeasible then
7:         Prune the node. Stop exploration on this branch
8:     else if $f(\boldsymbol{x}^*) <= f_{opt}$ then
9:         No better integer solution exists on this branch; prune node and stop exploration
10:    else if $\boldsymbol{x}^*$ is integral then
11:        $f_{opt} = f(\boldsymbol{x}^*)$
12:        $\boldsymbol{x}_{opt} = \boldsymbol{x}^*$
13:    else
14:        Select appropriate variable $x_i$ and create new bounds $(l^-, u^-)$ and $(l^+, u^+)$
15:        Add $MINLP(x_i, l^-, u^-)$ and $MINLP(x_i, l^+, u^+)$ to the heap.
16: return $f_{opt}, \boldsymbol{x}_{opt}$
```

2.2 Partial Differential Equations (PDEs)

Partial differential equations (PDEs) are a way of representing how quantities change with respect to continuous variables. For instance, it is necessary to use a PDE to measure how a variable such as pressure or heat changes with respect to both time and space. The PDE is defined over a domain, Ω, and involves the decision variables (three spatial dimensions and time), the (pressure or heat) function which returns a value given the configuration of the decision variables, and the k-order partial derivatives of this function with the respect to some or all of the decision variables. Formally, we may say that a system of partial differential equations for a function $\boldsymbol{u}(x_1, x_2, ..., x_n) = \boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}$ is of the form $\boldsymbol{F(x, u, u_x, u_{xx}, u_{xxx}, ...)}$, where $\boldsymbol{u_x} = \frac{\delta \boldsymbol{u}}{\delta x_1}, \frac{\delta \boldsymbol{u}}{\delta x_2}, ..., \frac{\delta \boldsymbol{u}}{\delta x_n}$, $\boldsymbol{u_{xx}} = \frac{\delta^2 \boldsymbol{u}}{\delta x_1 \delta x_1}, ..., \frac{\delta^2 \boldsymbol{u}}{\delta x_1 \delta x_n}$, and so on (i.e. the PDE is a function of a function $\boldsymbol{u}(x_1, ...x_n)$, variables $x_1, ..., x_n$, and the k-order ($k \in \mathbb{R}$) partial derivatives of $\boldsymbol{u}$ with respect to *at least two* variables). Note that if the derivative is only taken with respect to only one variable, this relationship is called an *ordinary* differential equation.

As an example, we may consider the Poisson Equation in three dimensions. The general form of Poisson's equation, (with Δ as the Laplacian) is $\Delta\varphi = f$, where we are given source term f and wish to find the value of φ over the domain Ω. If we consider this equation in the three dimensional plane, it reduces to:

$$(\frac{\delta^2}{\delta^2 x} + \frac{\delta^2}{\delta^2 y} + \frac{\delta^2}{\delta^2 z})\varphi(x, y, z) = f(x, y, z) \qquad x, y, z \in \Omega \tag{2.14}$$

As one can see, Equation 2.14 prescribes that the sum of partial second derivatives of φ with respect to each dimension, at any point in the 3 dimensional plane, must be equal to a predefined source term f, which could be a constant across the domain, or also a function depending on location and other parameters. One must note, however, that there are three regions of importance in a PDE. One is our domain Ω, another is the border of the domain $\delta\Omega$, which is often governed by a simple border condition, and the last is the region outside of Ω, where the PDE does not exist and therefore is not considered. Including an arbitrary border condition, we may write Equation 2.14 as (where $\mathcal{F}$ denotes the shorthand for the PDE):

$$\mathcal{F}(\varphi, x, y, z) = \begin{cases} (\frac{\delta^2}{\delta^2 x} + \frac{\delta^2}{\delta^2 y} + \frac{\delta^2}{\delta^2 z)}\varphi(x, y, z) = f(x, y, z) & x, y, z \in \Omega \\ (\frac{\delta^2}{\delta^2 x} + \frac{\delta^2}{\delta^2 y} + \frac{\delta^2}{\delta^2 z)}\varphi(x, y, z) = 0 & x, y, z \in \delta\Omega \end{cases} \tag{2.15}$$

While a full analysis of partial differential equations is out of the scope of this review, it is important to note the difficulty they pose, which motivates creative solution approximations. As PDEs are systems which occur over some domain Ω, the PDE models the relationship between the change in two or more variables *at each point* in the domain. Needless to say, in many situations, PDEs are not analytically tractable (especially over large domains or as the number of variables increases), and thus one must resort to approximation method to convert the variables of the PDE from existing in an infinite dimensional state space to a finite dimensional state space, after which the PDE may be solved as a massive (finite numbered) system of equations. In optimization theory, PDEs often appear as constraints; the majority of the effort of the optimization problem is then reduced to moulding the PDE constraint into a tractable form, rather than solving the actual problem.

2.2.1 PDEs and Discretization

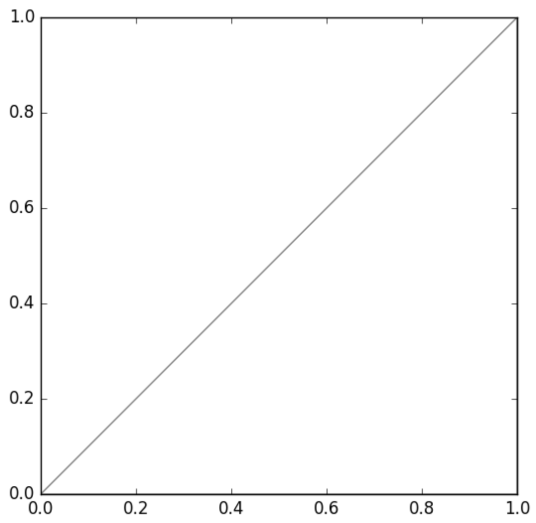
Consider the PDE as given in the constraint of Equation 2.3 (forgetting about the optimization problem for the moment). Inside of a continuous domain Ω, the PDE equals constant c, while $\boldsymbol{u}$ (a function of the vector of decision variables $\boldsymbol{x}$, the k-order partial derivatives of $\boldsymbol{u}$ with respect to $\boldsymbol{x}$, and function(s) which relate them) equal zero on the boundary of the domain.

$$\begin{cases} \mathcal{F}(\boldsymbol{u}, \boldsymbol{x}) = c \quad \boldsymbol{x}, \boldsymbol{u} \in \Omega \\ \boldsymbol{u} = 0 \quad \text{on } \delta\Omega \end{cases} \tag{2.16}$$

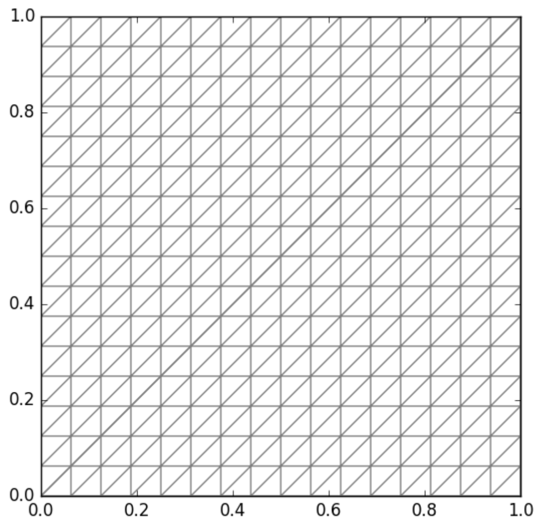
Given that PDEs are complex systems which vary over multiple dimensions (and PDEs can be infinitely-dimensional themselves), often, no closed-form solution is possible, or the computational demand of such a solution is infeasibly high, especially as the size of the domain Ω increases [6]. Consequently, one does not solve a PDE, but rather a tractable and discrete approximation of the continuous PDE. To do so, we first introduce a discrete domain known as a mesh, and then separate the domain of the continuous PDE into a host of sub-regions, approximating the PDE in each of those sub-regions using a combination of basis functions which exist in a *finite* vector space (we could use, for example, linear or polynomial basis functions). Importantly, this collection of piecewise functions is generated using a process which satisfies the boundary conditions of the PDE as well. Returning to the farm and fertilizer example (and dropping the time-varying assumption), consider the PDE that assigns a quality weight to fertilizer given its location on the farm, which is a square plot of land. This PDE is continuous, and thus the quality weight varies continuously its domain. If we suppose this relationship is modeled by a Poisson equation in two dimensions, we have:

$$\begin{cases} (\frac{\delta^2}{\delta^2 x} + \frac{\delta^2}{\delta^2 y})u(x, y) = f \quad x, y \in \Omega \\ u(x, y) = 0 \quad \text{on } \delta\Omega \end{cases} \tag{2.17}$$

This relationship tells us that the Laplacian of the PDE is equal to some source term f (say, a constant, $f = 5$) at each point in its continuous domain. As it is difficult to model this relationship over the continuous domain, the domain is broken up into sub-regions, each of which contains a function which approximates the PDE in that region. The domain thus becomes a



(a) A Continuous Domain - only one region

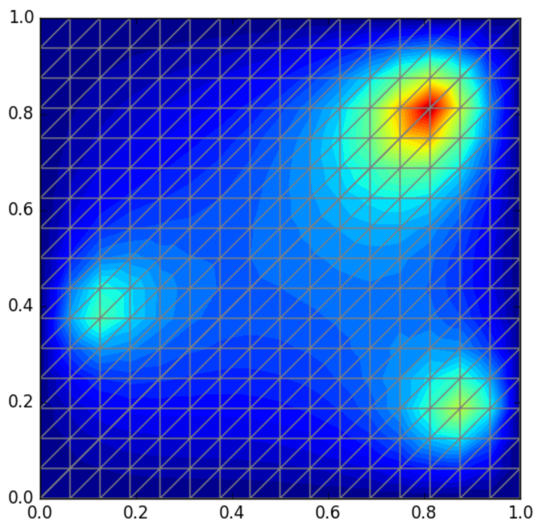


(b) A Discrete Domain - triangle sub-regions

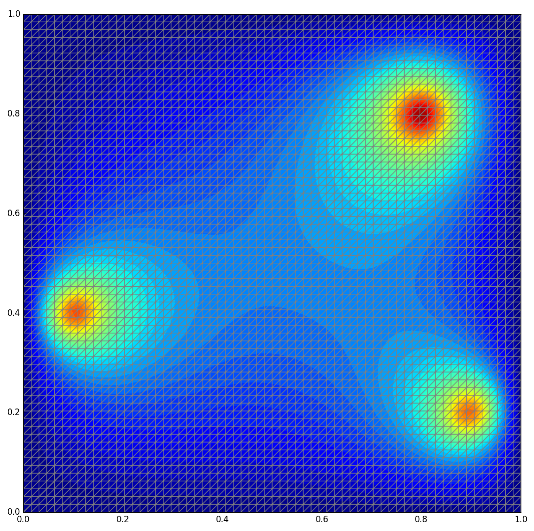
Figure 2.1: The continuous and discrete domains of a PDE

piecewise combination of approximations. Pictorally, we break up the continuous domain on the left into the piecewise domain on the right (each triangle becomes a sub-region):

One must note that the mesh can be refined in particularly important areas, and the functions which approximate the PDE in each sub-region can vary in order (linear, polynomial, etc.). The mesh size is an extremely important parameter of the PDE-discretization process as a coarser mesh is computationally cheap (a 16 x 16 mesh represents a PDE as values in a 16 x 16 matrix), but imprecise in its representation of the PDE, while a fine mesh is increasingly accurate, but computationally expensive to manage. A Continuous PDE would be represented by an infinitely fine mesh; naturally, this is computationally prohibitive as it would entail a k x k matrix to represent the PDE, with $k \to \infty$.



(a) A discretized PDE evaluated on a 16 x 16 mesh



(b) A discretized PDE evaluated on a 64 x 64 mesh

Figure 2.2: A PDE discretized on a coarse mesh (left) and fine mesh (right)

The actual PDE discretization is accomplished using one of three methods: the finite element method (which I will be using as it is easily implemented in the FEniCS framework), the finite volume method, or the finite difference method. The distinguishing characteristic of the finite element method (FEM) is that one can create a mesh over the domain of the PDE using any geometric shape to represent the sub-regions (triangles, squares, polygons, etc.), as exemplified by the triangular elements in Figure 2.2. The finite difference method (FDM) is another discretization scheme that is far easier to implement than FEM, but is also much less flexible and generally less accurate, as the FDM can only use cubes to create the mesh. Additionally, it has been noted that while it is liked for its simplicity, FDM can yield unrealistic results and does not necessarily conserve mass [17]. The finite volume method (FVM) is the last paradigm for discretizing PDEs, and will not be remarked upon further. In comparing the three methods, Oliver notes that these methods are not one-size-fits-all: the utility of each method is extremely dependent on the particular PDE being solved. To summarize, it is generally accepted that the finite element method is more broadly applicable to solving various PDEs, while the FDM and FVM are only the best method in particular scenarios; the ease of implementing the FDM/FVM, however, can offset the often superior, but difficult to implement FEM (though any sort of broad generalizations are inherently difficult to state in an unqualified manner). While I will give a short mathematical treatment of the FEM in the Appendix as applied to the case studies I am examining, I shall gloss over most mathematical details of discretization; the interested reader may refer to a mathematical overview and comparison of the three methods in Peiro and Sherwin's chapter of *Handbook of Materials Modeling* [18].

As a graphic comparison of methods, see Figure 2.3 below. As the continuous PDE whose domain is the complex mechanical region in Figure 2.3 is intractable, the domain is discretized using finite differencing and finite elements. Intuitively, the FDM approximation seems coarser and cruder, as it is hampered by solely using cubes to define each sub-region, while the polygonal finite elements are far more versatile.

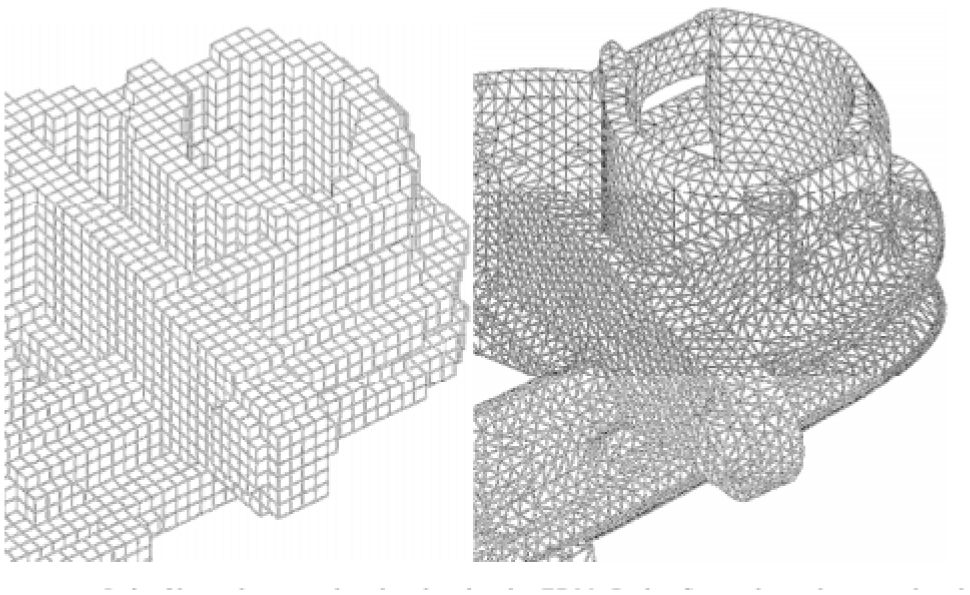


Figure 2.3: The finite difference method (left) and the finite element method (right) [19]

2.3 Mixed-Integer PDE-Constrained Optimization (MIPDECO)

MIPDECO problems can be written in their standard continuous form as in equation 2.3, where $\boldsymbol{x}$ and $\boldsymbol{u}$ exist in the appropriate and possibly infinitely-dimensional Hilbert Spaces. Some (or all) of the components in the decision variable vector $\boldsymbol{x}$ are required to be integers, and the decision variables are subject to equality and/or inequality constraints.

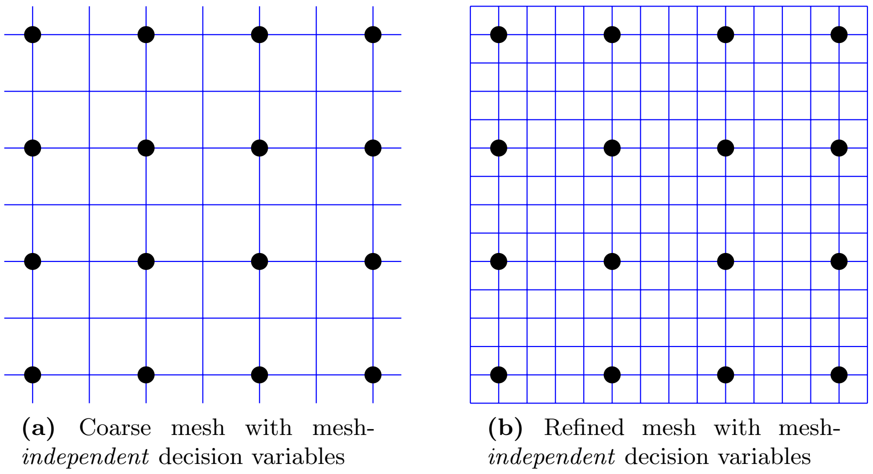
$$\min_{\boldsymbol{u},\boldsymbol{x}} \; J(\boldsymbol{u},\boldsymbol{x}) \quad s.t. \quad \begin{cases} \mathcal{F}(\boldsymbol{u},\boldsymbol{x}) = 0 \\ \boldsymbol{g}(\boldsymbol{x}) = 0 \\ \boldsymbol{h}(\boldsymbol{x}) \leq 0 \\ x_i \in \mathbb{Z} \quad \forall i \in 1,...,K \\ x_j \in \mathbb{R} \quad \forall j \in K+1,...,N \\ \boldsymbol{x},\boldsymbol{u} \in \mathcal{H}_1, \mathcal{H}_2 \end{cases} \tag{2.18}$$

2.3.1 The Exploding Mesh Problem

Before discretizing/optimizing a MIPDECO, we can observe that MIPDECO problems can be classified into two categories: those with mesh-independent decision variables, and those with mesh-dependent decision variables. Though both have the same general problem formulation, we now must consider the MIPDECO problem from 2.18 after continuous PDE $\mathcal{F}$ has been discretized. Denoting the discrete PDE as $\mathbb{F}$, and transforming our infinite dimensional Hilbert spaces into finite vector spaces $\mathcal{V}_1$ and $\mathcal{V}_2$, we have the discrete MIPDECO formulation:

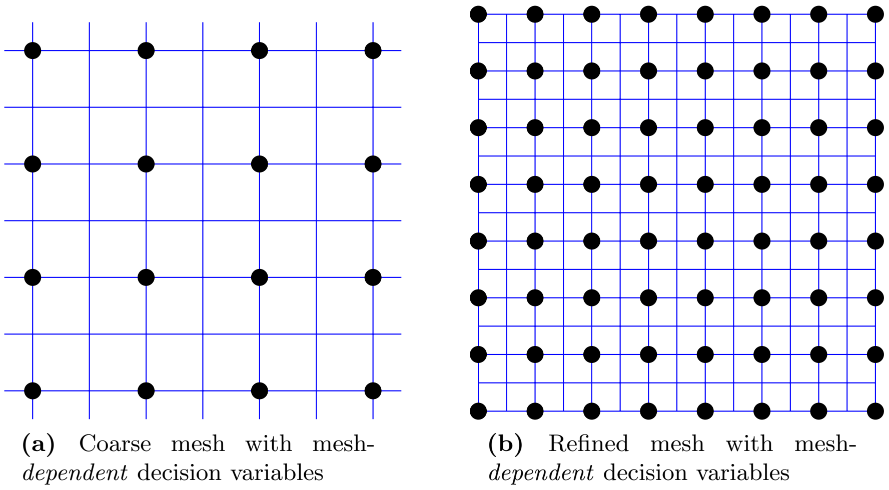
$$\min_{\boldsymbol{u},\boldsymbol{x}} \; J(\boldsymbol{u},\boldsymbol{x}) \quad s.t. \quad \begin{cases} \mathbb{F}(\boldsymbol{u},\boldsymbol{x}) = 0 \\ \boldsymbol{g}(\boldsymbol{x}) = 0 \\ \boldsymbol{h}(\boldsymbol{x}) \leq 0 \\ x_i \in \mathbb{Z} \quad \forall i \in 1,...,K \\ x_j \in \mathbb{R} \quad \forall j \in K+1,...,N \\ \boldsymbol{x},\boldsymbol{u} \in \mathcal{V}_1, \mathcal{V}_2 \end{cases} \tag{2.19}$$

Now if we consider $\mathbb{F}$ to model a relationship over a mesh such as in Figure 2.2, depending on the nature of the problem, the number of decision variables could stay constant whether we pick the 16 x 16 mesh or the 64 x 64 mesh as depicted in Figure 2.2. If this is the case, the decision variables are called mesh-*independent*. On the other hand, if the decision variables are mesh-*dependent*, the number of decision variables will increase as the domain of the discretized PDE is refined. Obviously, this distinction has significant ramifications on the scalability of a MIPDECO problem. Consider the situation below in which each black circle corresponds to a decision variable; in Figure 2.4, a square matrix of decision variables are embedded in the mesh. As the variables are mesh independent, the number of decision variables stays the same as the refinement increases. In Figure 2.5 the decision variables are mesh-dependent, and thus the number of decision variables scales quadratically with the mesh dimension. Consequently, with a m x m mesh any optimization algorithm with have memory complexity $\mathcal{O}(m^2)$ with regards to the decision variables, as opposed to $O(n)$ memory complexity for n mesh-independent decision variables in Figure 2.4, which will stay constant regardless of the value of m.


(a) Coarse mesh with mesh-*independent* decision variables

(b) Refined mesh with mesh-*independent* decision variables

Figure 2.4: Coarse (left) and refined (right) meshes with mesh-independent decision variables


(a) Coarse mesh with mesh-*dependent* decision variables

(b) Refined mesh with mesh-*dependent* decision variables

Figure 2.5: Coarse (left) and refined (right) meshes with mesh-dependent decision variables

Consequently, an optimization problem which tries to find the optimal values of the decision variables of the problem in Figure 2.4 will scale well in increasing mesh fineness. The problem represented by Figure 2.5, however, will rapidly become infeasible as the mesh fineness increases.

2.3.2 Alternatives in the MIPDECO Solution Process

Regardless of mesh-dependent or mesh-independent decision variables, there are two main approaches to solving PDE-constrained optimization problems such as the MIPDECO formulation in Equation 2.18. As aforementioned, the continuous form of the PDE must be discretized in order to make it computationally tractable (for details on PDE discretization and the mathematical process see the Appendix), using one of two methods. Given the initial MIPDECO problem with a continuous PDE constraint, one can either find the necessary optimality conditions (i.e. solve for the KKT conditions) and *then* discretize the PDE, yielding Solution 1 (Sol. 1) or one can first discretize the PDE constraint and then solve using nonlinear programming techniques (i.e. first order optimization methods), yielding Solution 2 (Sol. 2). Given these two techniques, one might ask if the answers are equivalent. The answer, unfortunately, is that the answers are not necessarily equal. That is, $\neg\Box(Sol.1 = Sol.2)$.

$$\begin{tikzcd}
\text{Beginning PDE} \arrow[r, "\text{Discretize}"] \arrow[d, "\text{Optimize}"] & \text{Discretized PDE} \arrow[r, "\text{Optimize NLP}"] & \text{Sol. 2} \arrow[d, "\text{Sol. 2}"] \\
\text{KKT/NC with cont. PDE} \arrow[r, "\text{Discretize KKT/NC}"'] & \text{Sol. 1} \arrow[r, "\text{Sol. 1}"'] & \neg\Box(Sol.1 = Sol.2)
\end{tikzcd}$$

Figure 2.6: Discretization order matters: discretize then optimize vs. optimize then discretize

Unsatisfyingly, it also appears that neither approach is objectively better, but rather the optimality of each approach is determined by the specific problem one is attempting to solve (much like discretization methods themselves) [21]. A good comparison of the techniques is given by Haber:

> In the Discretize then Optimize (DO) we may need to differentiate computational facilitators such as mesh generation routines. This can become rather complicated (and may not be differentiable). On the other hand, the Optimize and Discretize (OD) approach we discretize the necessary conditions (the derivatives of the Lagrangian). The discretized necessary conditions may not be a true gradient of any objective function and thus, if the mesh is not sufficiently fine, can lead to the wrong descent direction [20].

Note that in the following mathematical details and my own implementation, I will be using the Discretize-then-Optimize approach, which is the standard method in the limited corpus of MIPDECO case studies.

2.3.3 Solving a Generic MIPDECO Problem

Combining the previous sections, we can codify a general method of formulating and solving a MIPDECO problem. Note that this a general and high-level algorithm and that each of the steps will be carefully picked apart in the following chapters applied to specific MIPDECO problems (with the exception of step 3 which is relegated to the Appendix); moreover, note that the choice of error metric is highly dependent upon the time of problem.

Algorithm 6 A high level MIPDECO solution algorithm

```
1: Identify the MIPDECO Problem, constraints, and error metric
2: Mathematically define the objective function, equality and inequality constraints, continuous
   and integer decision variables, the system of PDEs which governs the physics of the problem,
   and the domain, $\Omega$, over which these variables interact.
3: Discretize the PDE (see Appendix), creating the finite approximation of the true PDE in $\Omega$
4: (Optional) Eliminate the PDE from the optimization problem using algebraic manipulation
5: Solve the (reduced) optimization problem exactly using MINLP methods (branch and cut)
   or approximately using first-order methods (gradient descent on continuous relaxation of
   integer variables).
6: Measure error metric and analyze computational time/memory complexity as required
```

Chapter 3

The Source Inversion Problem

3.1 The Problem Layout

Following Leyffer et al. [23], I will solve the Source Inversion MIPDECO problem, though with the use of adjoint-based gradient optimization methods (implemented with the FEniCS PDE-solving framework and dolfin-adjoint), instead of the MINLP approach. For a comprehensive mathematical explanation and justification of adjoint-based gradient optimization methods, see the Appendix. Naturally, using gradients will require lifting or modification of the integer constraints (done *after* the discretization of the PDE); integer solutions will be obtained by using a rounding heuristic, as well as through the implementation of a penalty algorithm.

The Source Inversion problem is a function-tracking model which attempts to minimize the distance between the solution to a reference PDE $\bar{\boldsymbol{u}}$ and the solution to a source PDE $\boldsymbol{u}$ (which is a Poisson equation with the source term equalling the sum of a host of source functions) over the simple box domain Ω, paired with a Dirichlet boundary condition. This minimization problem also requires that the maximum number of activated source functions does not exceed a user-defined integer S, and that each source function at location kl is modeled as a Gaussian function f_{kl} with binary coefficient w_{kl} (which controls the activation/inactivation of each source function). In other words, we want to find the most accurate way to recreate the solution to the reference PDE with a combination of up to S source functions.

$$\min_{\boldsymbol{u},\boldsymbol{w}} \mathcal{J}(\boldsymbol{u},\boldsymbol{w}) = \frac{1}{2}\int_{\Omega}(\boldsymbol{u}-\bar{\boldsymbol{u}})^2 d\Omega \quad s.t. \quad \begin{cases} -\Delta\boldsymbol{u} = \sum_{k,l} w_{kl} f_{kl} \quad \in \Omega \\ \boldsymbol{u} = 0 \quad \in \delta\Omega \\ \sum_{k,l} w_{kl} \leq S \\ w_{kl} \in \{0,1\} \;\; \forall k,l \end{cases} \tag{3.1}$$

At a high level, we can imagine the problem as such: first, we create a mesh (a 2-dimensional grid that represents the box Ω) and define a finite function space (linear, polynomial, etc. basis functions), elements of which will approximate the PDE over this mesh. As aforementioned, I am using Gaussian reference (and source) functions, so that $f_{kl}(x,y) \equiv a \cdot \exp(\frac{-((x_k-x)^2+(y_l-y)^2)}{\sigma})$, with a constant a (which is akin to a dirac pulse), and variance $\sigma > 0$. Each of these terms is centered at a point in the mesh, and has some of its density spread over all other vertices.

Given our mesh and function space, we now create a reference function (with $\mathbb{V}$ = vertices of the mesh) for given x_i, y_j:

$$\bar{f_{ij}} = a \cdot \exp(\frac{-((x_i-x)^2+(y_j-y)^2)}{\sigma}) \quad \in \Omega; \quad i,j \notin \mathbb{V}(mesh) \tag{3.2}$$

Eliding the mathematical details (see the Appendix), the finite element method finds the solution to the discretized reference PDE $\bar{\boldsymbol{u}}^d$ such that (v = test function, $\mathcal{V}$ = finite Hilbert space):

$$\int_\Omega \langle \nabla \bar{\boldsymbol{u}}^d, \nabla v \rangle \mathrm{d}x = \int_\Omega \bar{f_{ij}} v \ \mathrm{d}x \quad \forall v \in \mathcal{V} \tag{3.3}$$

Given this reference approximation, we now must create n^2 source terms (also Gaussian bumps) to embed within the larger m x m mesh ($n < m$). It is important to note that these terms are created as a square submatrix so that the functions are evenly spaced on the mesh, with each function centered on a mesh vertex. Given a field of n^2 of such sources, we must sum all of the sources, each of which is multiplied by a control variable w_{kl}. We then solve the weak formulation of the Poisson equation depicted below (see Appendix) using FEniCS, and denote the solution to this discretized PDE $\boldsymbol{u}^d$.

$$\int_\Omega \langle \nabla \boldsymbol{u}^d, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (w_{kl} \cdot \int_\Omega f_{kl} \cdot v \ \mathrm{d}x) \quad \forall v \in \mathcal{V} \tag{3.4}$$

After solving for $\bar{\boldsymbol{u}}^d$ and $\boldsymbol{u}^d$ – a task that is done automatically in the FEniCS framework, the optimization problem can be transformed from a problem existing in continuous space over Ω to a problem which exists in the discrete domain Θ. Thus we can reformulate our problem (with $\boldsymbol{u}^d = \boldsymbol{u}^d(\boldsymbol{w})$ and $\bar{\boldsymbol{u}}^d$ being the finite vectors approximating the integral) as:

$$\arg\min_{\boldsymbol{w}} \ \mathcal{J}(\boldsymbol{w}) = \frac{1}{2} || \boldsymbol{u}^d(\boldsymbol{w}) - \bar{\boldsymbol{u}}^d ||_2^2 \quad s.t. \quad \begin{cases} \sum_{k,l} w_{kl} \leq S \\ w_{kl} \in \{0,1\} \ \forall k,l \\ \boldsymbol{u}, \boldsymbol{w} \in \Theta \end{cases} \tag{3.5}$$

Given Equation 3.5, we have a few options. We can easily solve the continuous relaxation of the problem; this is achieved using the sequential least squares quadratic programming (SQP) algorithm as implemented in SciPy. Yet the binary integer constraints are the troublemaker in this problem, rendering gradient-based techniques unusable (i.e. maintaining the constraints necessitates employing MINLP algorithms such as branch and bound). In order to solve this issue, we may add a penalty function to the objective functional in order to penalize non-binary w_{kl} values and extract a natural integer solution. Thus we change Equation 3.5 (eliding the vector space for simplicity) to:

$$\arg\min_{\boldsymbol{w}} \ \mathcal{L}(\boldsymbol{w}) = \frac{1}{2} || \boldsymbol{u}^d(\boldsymbol{w}) - \bar{\boldsymbol{u}}^d ||_2^2 + \alpha \sum_{k,l} w_{kl}(1 - w_{kl}) \quad s.t. \quad \sum_{k,l} w_{kl} \leq S \tag{3.6}$$

As mentioned in the previous chapter, the parameter α must be tuned to properly penalize non-binary values of w_{kl} (also note that I am using a single α, as tuning α_{kl} for each source function would require far too much time). If α is too large, then the penalty term will change the optimization problem so that the objective function is minimized just by finding a feasible, rather than optimal, solution. If the α is too low, then the optimal solution will increasingly approach the optimal continuous solution. Finding this optimal α^* is an inexact task, but ideally the optimizing algorithm is run repeatedly starting with $\alpha = 0$, and increasing α in tiny increments until an integer solution (or a solution that is integer within an accepted error tolerance) is found. It is important to note that the bounds on w_{kl}, $0 \leq w_{kl} \leq 1$ have been removed due to numerical instability. While keeping the bounds in some situations causes no issue, in a few of my tests, the bounds have prevented a proper integer solution from being found – see Sager et al. [25] for an in-depth explanation of the issue applied to a mixed-integer control problem.

In order to solve this problem in any configuration and with any algorithm, the gradient of the objective functional must be taken during each iteration. Depending on the dimensions of the each component of the optimization problem, (i.e. $\mathcal{J}, \boldsymbol{u}, \boldsymbol{w}$), this gradient evaluation can be computationally taxing. As this project is built using dolfin-adjoint (which also wraps around SciPy), it is noted that any optimization algorithm *automatically uses the adjoint approach* in finding the gradient during each iteration. This massively reduces the computational complexity due to the inherent characteristics of PDE-constrained optimization [24]. A mathematical overview of adjoints and their utility when applied to PDE-constrained optimization is available in the Appendix.

3.2 Algorithms and Scalability

As aforementioned, due to the fact that $w_{kl} \in \{0, 1\}$, I either must use a branch and bound to preserve perfect accuracy, or use approximate gradient-based methods. As I am attempting to build a scalable solution, and MINLP solvers are often computationally infeasible for real-world applications (they produce enormous, albeit accurate, trees), I will test a heuristic gradient based algorithm which rounds the continuous solution to an integer solution, a gradient-based algorithm which incorporates a penalty function to yield an automatic integer solution (as above in 3.6), and an algorithm which uses the continuous solution as inputs to the penalty algorithm for another automatic integer solution. All of these are compared against a MINLP branch and bound solver implemented in CasADi. Allowing n to be the number of w_{kl}'s and m the number of elements in the mesh, The key differences between the tested algorithms are as follows:

Table 3.1: Characteristics of Algorithms Solving the Source Inversion Problem

MINLP	Heuristic	Penalty	Two-Step
Exact Solution	Rounded Solution	Approximate Solution	Approximate Solution
$\sim$ scalable in n	$\sim$ scalable in n	$\sim$ scalable in n	$\sim$ scalable in n
$\sim$ scalable in m	scalable in m	scalable in m	scalable in m

As often happens, the algorithms boil down to efficiency vis-a-vis accuracy. While the MINLP approach ensures a precise solution through near-exhaustive search, the size (the total number of variables $n+m$) of the discetized PDE can be prohibitive, and we must resort to less accurate, but computationally feasible, gradient-based heuristic, penalty, and two-step algorithms.

3.2.1 The Continuous Relaxation: an Alternate Route

In this MIPDECO problem, we are attempting to ascertain the optimal $\boldsymbol{w}$ which minimizes the distance between the reference solution and the source solution. One must note, however, that the computational complexity of this optimization problem necessarily increases in $\boldsymbol{w}$. Though the number of source functions is independent of the mesh size (hence we do not run into the exploding-mesh problem whereby the number of decision variables necessary increase in increasing mesh refinement), any optimization algorithm solving the aforementioned formulations of the Source Inversion problem will scale poorly in $\boldsymbol{w}$, and thus the number of source functions embedded in the mesh will limit the solvability of the problem. Intuitively, one might think of increasing the number of source functions as increasing the number of controls in the problem, all of which must be optimized. Though there exists an easy remedy to this scaling issue, it is only applicable in the continuous relaxation of this problem.

First, we can remove $\boldsymbol{w}$ from the problem; when we solve for $\boldsymbol{u}^d$ as in Equation 3.4, we instead solve for $\boldsymbol{u}^d$ as the solution to:

$$\int_\Omega \langle \nabla \boldsymbol{u}^d, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (\int_\Omega f_{kl} \cdot v \ \mathrm{d}x) \quad \forall v \in \mathcal{V} \tag{3.7}$$

Due to the fact that the $\boldsymbol{w}$ vector of coefficients is missing from the optimization problem, one can create a constraint matrix $\boldsymbol{A}$ in order to implicitly extract the optimal w_{kl} values in the following step. The constraint matrix is created as follows: if we allow f_{kl}^{ab} to denote the value of the source function centered at mesh vertex (k, l) and evaluated at mesh vertex (a, b), then we can create a row vector which contains the value of the given source function at every single point on the mesh. That is, $\boldsymbol{v_{kl}} = \begin{bmatrix} f_{kl}^{11} & f_{kl}^{12} & f_{kl}^{13} ... & ...f_{kl}^{kl} & f_{kl}^{k,l+1} ... f_{kl}^{mm} \end{bmatrix}$. By constructing such a vector for all n^2 source functions, the constraint matrix is created by taking the transpose of each vector, and stacking these column vectors next to each other so that $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{v_{11}^T} & \boldsymbol{v_{12}^T} ... & ...\boldsymbol{v_{kl}^T} ... & ...\boldsymbol{v_{ll}^T} \end{bmatrix}$. This constraint matrix $\boldsymbol{A}$ is a m^2 x n^2 matrix, with n^2 different source functions evaluated at each of the m^2 points on the mesh.

Given this constraint matrix, and solving 3.7 for $\boldsymbol{u}^d$, one can reform the optimization problem in 3.5 by formulating $\boldsymbol{u}$ as a function of the sum of all source functions over the grid. In other words, $\boldsymbol{u} = \boldsymbol{g}(\sum_{kl} f_{kl})$, and instead of using $\boldsymbol{w}$ as a control (where increasing the number of source functions increases the number of controls), one uses $\sum_{kl} f_{kl}$ as the control. The crucial point is that this sum is always a single control variable, and thus adding or subtracting source functions has no effect on the actual number of controls in the optimization algorithms. The new optimization problem can be written as:

$$\underset{\sum_{k,l} f_{kl}}{\arg\min} \ \mathcal{J}(\sum_{k,l} f_{kl}) = \frac{1}{2} ||\boldsymbol{u}^d(\sum_{k,l} f_{kl}) - \bar{\boldsymbol{u}}^d||_2^2 \tag{3.8}$$

This unconstrained optimization problem is easily solved using BFGS in SciPy. It is important to note that this problem does not solve for $\boldsymbol{w}$, but rather *the optimal value of the sum of source functions at each vertex in the mesh.* Reshaping the output of this optimization routine into a m^2 x 1 vector $\boldsymbol{b}$, one then can form a second optimization problem in order to extract the optimal value of $\boldsymbol{w}$, which attempts to minimize the difference between the actual value and optimal value of the sum of source functions at each vertex, given a vector of decision variables $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}} \left\| \begin{bmatrix} f_{11}^{11} & f_{12}^{11} & ... & ... & f_{ll}^{11} \\ f_{11}^{12} & f_{12}^{12} & ... & ... & f_{ll}^{12} \\ . & & & & \\ . & & & & \\ f_{11}^{mm} & f_{12}^{mm} & ... & ... & f_{ll}^{mm} \end{bmatrix} \cdot \begin{bmatrix} w_{11} \\ w_{12} \\ . \\ . \\ w_{ll} \end{bmatrix} - \begin{bmatrix} b_{11} \\ b_{12} \\ . \\ . \\ b_{mm} \end{bmatrix} \right\|_2^2 \tag{3.9}$$

This can be concisely written (recognizing that $\boldsymbol{A}$ is the constraint matrix):

$$\min_{w} \ \mathcal{L} = \langle \boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}, \boldsymbol{A}\boldsymbol{w} - \boldsymbol{b} \rangle \quad s.t. \quad 0 \leq w_i \leq 1 \ \forall i \tag{3.10}$$

One may notice that this is simply an application of Ordinary Least Squares (albeit with a constraint), whose convexity in $\boldsymbol{w}$ is a well-known result. Thus we can be assured that we will find the optimal global solution for the coefficients $\boldsymbol{w}$, *given an optimal* $\boldsymbol{b}$. As a short proof of convexity:

$$\frac{\delta \mathcal{L}}{\delta \boldsymbol{w}} = 2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{w} - 2\boldsymbol{A}\boldsymbol{b} \implies \frac{\delta^2 \mathcal{L}}{\delta^2 \boldsymbol{w}} = 2\boldsymbol{A}^T\boldsymbol{A} \succ 0 \tag{3.11}$$

The resulting Hessian is of full rank by construction of $\boldsymbol{A}$ and thus is strictly positive definite $\forall \boldsymbol{d} \in \Omega$.

After this long digression, we now arrive at the catch. While this technique might yield the continuous solutions faster than algorithms which use $\boldsymbol{w}$ as a control (for a large number of source functions), this method has no way of solving for integer values of $\boldsymbol{w}$, as $\boldsymbol{w}$ is never directly solved for. Compounding on this issue is the fact that convexity only holds assuming the first-stage optimization problem returns the optimal value of $\boldsymbol{b}$, of which there is no guarantee.

In the sections below, I present several algorithms which us $\boldsymbol{w}$ as the control variable. First, I present a continuous algorithm which rounds the largest S w_{kl} parameters as found in the continuous solution to 1, and rounds all others to 0. The penalty algorithm is presented next, which uses a penalty function to penalize non-binary values of w_{kl}, eventually yielding an optimal integer solution without any sort of heuristic method of morphing a continuous solution into an integer solution. Finally, the two algorithms are combined so that the optimal continuous vector $\boldsymbol{w}^*_{\boldsymbol{cont}}$ becomes the initialization vector $\boldsymbol{w_{init}}$ of the penalty algorithm.

3.3 The Heuristic Integer Algorithm

The heuristic integer algorithm takes the continuous relaxation of the w_{kl} binary variables and then rounds each continuous w_{kl} either up or down depending on its optimal continuous magnitude and the value of S. Note that the optimal continuous values may be obtained either using the continuous relaxation of Equation 3.5, or the entirely different least-squares process as described in the previous subsection. In practice, the former method was used.

Algorithm 7 Continuous (w control)/Heuristic Integer Algorithm (Source Inversion)

```
1: Choose mesh size $m$, number of sources $n^2$, and activated source bound $S$.
2: Create $\Theta$, a *discrete* $m$ x $m$ mesh (the discrete domain of the PDE)
3: Choose $k$, the order of the function space in which the the mesh, $\boldsymbol{u}$, $\bar{\boldsymbol{u}}$ exist
4: Create the $k$-order function space $V$
5: Create Gaussian reference function $\bar{f}$
6: Find reference PDE $\bar{\boldsymbol{u}}^d$, the solution to $\int_\Theta \langle \nabla \bar{\boldsymbol{u}}^d, \nabla v \rangle \mathrm{d}x = \int_\Theta \bar{f} v \, \mathrm{d}x \;\; \forall v \in V$
7: Create $n^2$ evenly spaced source functions and constant objects with $w_{kl}^{init} = 1 \quad \forall k, l$
8: Project the sum of the source terms $\sum_{k,l} w_{kl}^{init} f_{kl}$ into $\Theta$
9: Find source PDE $\boldsymbol{u}^d$, the solution to: $\int_\Theta \langle \nabla \boldsymbol{u}^d, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (\int_\Theta w_{kl}^{init} \cdot f_{kl} \cdot v \, \mathrm{d}x) \;\; \forall v \in V$
10: Form the functional $J(\boldsymbol{u}^d) = (\frac{1}{2} \bar{\boldsymbol{u}}^d - \boldsymbol{u}^d)^2$ and make $\boldsymbol{w}$ into a control object
11: Form reduced functional $\hat{J}(\boldsymbol{u}(\boldsymbol{w}))$ and create bounds such that $0 \le w_{kl} \le 1$
12: Optimise the reduced functional $\min_{\boldsymbol{w}} \; \hat{\boldsymbol{J}}(\boldsymbol{w})$ with bounds using SLSQP in Scipy
13: Step 12 returns $\boldsymbol{w}^*$, the optimal continuous values of $\boldsymbol{w}$.
14: Create $\boldsymbol{w^{int}}$ by rounding $S$ largest elements of $\boldsymbol{w}^*$ to 1 and setting all others to 0
15: Find recon. PDE $\boldsymbol{u}^r$, as solution to $\int_\Theta \langle \nabla \boldsymbol{u}^r, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (w_{kl}^{int} \cdot \int_\Theta f_{kl} \cdot v \, \mathrm{d}x) \;\; \forall v \in V$
16: Assess the difference between $\boldsymbol{u}^r$, the reconstruction of the reference PDE given the optimal
    integer (resp. continuous) values of $\boldsymbol{w^{int}}$ for the Heuristic Integer Algorithm (resp. $\boldsymbol{w}^*$ for
    the Continuous (w control) Algorithm) and reference solution $\bar{\boldsymbol{u}}$
```

3.4 The Penalty Algorithm

The penalty algorithm proceeds in a similar manner as the heuristic algorithm, but diverges at step 10, where the objective functional is augmented with a term which penalizes non-binary values of w_{kl}. With a properly chosen α, this algorithm returns integer values. Note that this algorithm assumes a pre-selected value of α^*.

Algorithm 8 Penalty Algorithm (Source Inversion)

```
 1: Choose mesh size $m$, number of sources $n^2$, and activated source bound $S$.
 2: Create $\Theta$, a *discrete* $m$ x $m$ mesh (the discrete domain of the PDE)
 3: Choose $k$, the order of the function space in which the the mesh, $\boldsymbol{u}$, $\bar{\boldsymbol{u}}$ exist
 4: Create the $k$-order function space $V$
 5: Create Gaussian reference function $\bar{f}$
 6: Find reference PDE $\bar{\boldsymbol{u}}^d$, the solution to $\int_\Theta \langle \nabla \bar{\boldsymbol{u}}^d, \nabla v \rangle \mathrm{d}x = \int_\Theta \bar{f} v \ \mathrm{d}x \ \ \forall v \in V$
 7: Create $n^2$ evenly spaced source functions and constant objects with $w_{kl}^{init} = 1 \quad \forall k, l$
 8: Project the sum of the source terms $\sum_{k,l} w_{kl}^{init} f_{kl}$ into $\Theta$
 9: Find source PDE $\boldsymbol{u}^d$, the solution to: $\int_\Theta \langle \nabla \boldsymbol{u}^d, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (\int_\Theta w_{kl}^{init} \cdot f_{kl} \cdot v \ \mathrm{d}x) \ \ \forall v \in V$
10: Form the functional $J(\boldsymbol{w}, \boldsymbol{u}^d) = \frac{1}{2} (\bar{\boldsymbol{u}}^d - \boldsymbol{u}^d)^2 + \alpha^* \sum_{k,l} |w_{kl}(1 - w_{kl})|$
11: Form reduced functional $\hat{J}(\boldsymbol{u(w)})$ and create bounds such that $0 \leq w_{kl} \leq 1$
12: Solve reduced functional for $\boldsymbol{w^*} = \arg\min_{\boldsymbol{w}} \ \boldsymbol{\hat{J}(w)}$ with bounds using SLSQP in Scipy
13: Step 12 returns $\boldsymbol{w^*}$, the optimal (hopefully integer) values of $\boldsymbol{w}$
14: Find recon. PDE $\boldsymbol{u}^r$, as solution to $\int_\Theta \langle \nabla \boldsymbol{u}^r, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (w_{kl}^{int} \cdot \int_\Theta f_{kl} \cdot v \ \mathrm{d}x) \ \ \forall v \in V$
15: Assess the difference between $\boldsymbol{u}^r$, the reconstruction of the reference PDE given the optimal
    integer values of $\boldsymbol{w^{int}}$, and the reference solution $\bar{\boldsymbol{u}}$
```

Naturally, we must find the proper α^* in order to run the above algorithm for any configuration of n and m^2. In practice, this has been obtained by running the algorithm a number of times, starting with $\alpha = .001$ or $.0001$ depending on the number of source functions, and multiplying by step size k when the algorithm does not penalize the objective functional enough while reducing k by $\frac{1}{2}$ if the algorithm returns a $\boldsymbol{w^*}$ that is uniformly shrinking towards zero. One must note that in this formulation of the Source Inversion problem, the reference function is a linear combination of Gaussians (suppose that there are T individual Gaussians). To preserve scale (as the hyperparameters of the Gaussians are equal across source and reference functions), we know that the integer solution will correspond to exactly T source functions being activated. Thus we can form a function that automatically finds α by changing α in response to the sum of the optimal $\boldsymbol{w}$, which, in an integer setting, will equal T.

Algorithm 9 Finding the optimal penalty term α (Source Inversion)

```
 1: Choose $\alpha_{init} = .001$, $k = 5$, $n$, $m$, $S$. Choose error tolerance $\epsilon$. Set $\boldsymbol{w}$ to a vector of 1s.
 2: Define $T$ as the number of individual Gaussians which make up the reference function
 3: while $||\boldsymbol{w}||_2^2 > S$ or $||\boldsymbol{w}||_2^2 < T - \epsilon$ or $\lfloor ||\boldsymbol{w}||_2^2 \rfloor - ||\boldsymbol{w}||_2^2 > \epsilon$ do
 4:     $\boldsymbol{w_{opt}}$ = output of Penalty Function Algorithm($\alpha$)
 5:     if $||\boldsymbol{w}||_2^2 > T + \epsilon$ then
 6:         $\alpha = \alpha \cdot k$
 7:     else
 8:         $\alpha = \alpha / k$
 9:         $k = k/2$
10:         if $k < 1$ then
11:             $k = 1 + .5 \cdot (2 \cdot k - 1)$
12:         $\alpha = \alpha \cdot k$
13: return $\alpha^* = \alpha$
```

3.5 The Two-Step Algorithm

The two-step algorithm combines the continuous (w control) and the penalty algorithms into a single pipeline, following the work of Huang and Wang. In their paper examining a novel method to efficiently solve integer linear programs, they formulate a two-step algorithm in which they first reformulate all binary constraints into continuous constraints $w_i \in \{0, 1\} \rightarrow 0 \leq w_i \leq 1$, and then solve the relaxed LP using the interior point/revised simplex algorithm in order to find a suboptimal but good starting point for step 2. In step 2, the binary constraints are again reformulated as $w_i \in \{0, 1\} \rightarrow w_i(w_i - 1) = 0$, after which they can be incorporated into a Lagrangian objective functional which can be optimized according to any gradient based method [26]. Applying this methodology to the Source Inversion problem, we first solve for the continuous solution, and then use the optimal continuous values as the inputs (i.e. the w_{kl}^{init}) for the penalty algorithm. While Huang and Wang used the interior point method to solve the first stage algorithm, I have decided to stick with SQP in SciPy in order to maintain a consistency across all of the gradient algorithms instead of using IPOPT for step 1 in the two-step algorithm and SQP for step 2.

Algorithm 10 Two-Step Algorithm (Source Inversion)

```
 1: Run steps 1-12 of the Continuous (w control) algorithm
 2: Step 12 returns $\boldsymbol{w^*}$, the optimal continuous values of $\boldsymbol{w}$
 3: Create $n^2$ evenly spaced source functions and constant objects with $w_{kl}^{init} = w_{kl}^* \quad \forall k, l$
 4: Project the sum of the source terms $\sum_{k,l} w_{kl}^* f_{kl}$ into $\Theta$
 5: Find source PDE $\boldsymbol{u}^d$, the solution to: $\int_\Theta \langle \nabla \boldsymbol{u}^d, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (\int_\Theta w_{kl}^* \cdot f_{kl} \cdot v \, \mathrm{d}x) \ \ \forall v \in V$
 6: Form the functional $J(\boldsymbol{w}, \boldsymbol{u}^d) = \frac{1}{2} ||\bar{\boldsymbol{u}}^d - \boldsymbol{u}^d||_2^2 + \alpha \sum_{k,l} w_{kl}^* (w_{kl}^* - 1)$
 7: Form reduced functional $\hat{J}(\boldsymbol{u(w^*)})$ and create bounds such that $0 \leq w_{kl} \leq 1$
 8: Find $\boldsymbol{w}^{int} = \underset{\boldsymbol{w}}{\arg\min} \ \hat{J}\boldsymbol{(w^*)}$ with bounds using SLSQP in Scipy
 9: Step 8 returns $\boldsymbol{w}^{int}$, the optimal integer values of $\boldsymbol{w}$
10: Find recon. PDE $\boldsymbol{u}^r$, as solution to $\int_\Theta \langle \nabla \boldsymbol{u}^r, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (w_{kl}^{int} \cdot \int_\Theta f_{kl} \cdot v \, \mathrm{d}x) \ \ \forall v \in V$
11: Assess the difference between $\boldsymbol{u}^r$, the reconstruction of the reference PDE given the optimal
    integer values of $\boldsymbol{w}^{int}$, and the reference solution $\bar{\boldsymbol{u}}$
```

3.6 The Branch and Bound Algorithm

The MINLP algorithm proceeds in a similar way to the aforementioned algorithms. In particular, everything is identical until the optimization step. Instead of using SQP or optimizing with a penalty functions, the MINLP algorithm uses branch and bound with IPOPT as the subproblem solver to find exact integer solutions.

Algorithm 11 MINLP Algorithm (Source Inversion)

```
1: Run Steps 1-11 of the Continuous (w control) algorithm
2: Optimise the reduced functional $\min_{\boldsymbol{w}} \hat{\boldsymbol{J}}(\boldsymbol{w})$ using branch and bound implemented with CasADi and IPOPT
3: Step 2 returns $\boldsymbol{w}^{bnb}$, the optimal integer values of $\boldsymbol{w}$
4: Find recon. PDE $\boldsymbol{u}^r$, as solution to $\int_\Theta \langle \nabla \boldsymbol{u}^r, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (w_{kl}^{bnb} \cdot \int_\Theta f_{kl} \cdot v \, \mathrm{d}x) \;\; \forall v \in V$
5: Assess the difference between $\boldsymbol{u}^r$, the reconstruction of the reference PDE given the optimal integer values of $\boldsymbol{w}^{bnb}$, and the reference solution $\bar{\boldsymbol{u}}$
```

3.7 A Source Inversion Error Metric

In this problem, the natural source of error comes from the difference between the solution to the reference PDE and the solution to the reconstructed reference PDE (that is, $\boldsymbol{u}^r$ that is found in steps 14, 15, 18, and 5 respectively, in each algorithm). One must note, however, that the absolute and per-grid-point difference is highly dependent on the size of the mesh. Thus it is important to scale the error at each point in order to compare the errors across mesh sizes and algorithms. As the error became quite small and was always less than 1 at each mesh vertex, I used an $L1$ error metric which measures the average per-grid-point reconstruction error, scaled my mesh size:

$$\epsilon_{L1} = \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} |\bar{u}_{ij} - u_{ij}^r| \tag{3.12}$$

Chapter 4

Solving the Source Inversion Problem

4.1 The Mesh and Reference Functions

In order to test algorithms across a variety of parameter configurations, I used a standard reference function defined on a two-dimensional unit square mesh. Though the refinement of the mesh varied across parameter configurations, each axis always had unit length (i.e. $0 \leq x \leq 1$ and $0 \leq y \leq 1$). For a 16 x 16 mesh, a unit square was broken up to have vertices in intervals of .0625; on a 32 x 32 mesh, the vertices of the mesh are in intervals of .01325, and so on. In my testing, I used the mesh sizes 16 x 16, 32 x 32, 64 x 64, and 128 x 128 (to illustrate time complexity, I also tried out a few other mesh sizes, including $m = 96, m = 200, m = 300$).

The reference function was identical in all situations, being a linear combination of 3 Gaussian functions, each of which having values of a and σ which are identical to those of the source functions.

$$\bar{f} = a \cdot (e^{-\sigma^{-1}((x-.1)^2+(y-.4)^2)} + e^{-\sigma^{-1}((x-.8)^2+(y-.8)^2)} + e^{-\sigma^{-1}((x-.9)^2+(y-.2)^2)}) \tag{4.1}$$

The centroid of each Gaussian component of the reference function was picked soas to *not* coincide with any mesh vertex, regardless of which mesh size ($m = 16$, $m = 32$, $m = 64$, etc...) was used, ensuring that no source functions and Gaussian components of the reference function share exactly the same footprint.

As aforementioned, the continuous reference function was discretized with the finite element method, which works by finding a discrete vector $\bar{\boldsymbol{u}}^d$ such that (v = test function, $\mathcal{V}$ = finite Hilbert space):

$$\int_\Omega \langle \nabla \bar{\boldsymbol{u}}^d, \nabla v \rangle \mathrm{d}x = \int_\Omega \bar{f} v \ \ \mathrm{d}x \quad \forall v \in \mathcal{V} \tag{4.2}$$

Figure 4.1 below depicts both the initial reference function $\bar{f}$ and the discretized solution to the reference PDE $\bar{\boldsymbol{u}}^d$. While the image below corresponds to a 32 x 32 mesh, the peaks are as one would imagine with the other mesh sizes – coarser Gaussians and a coarser discrete reference solution for a 16 x 16 mesh, and more refined versions of the below image for a 64 x 64 mesh.

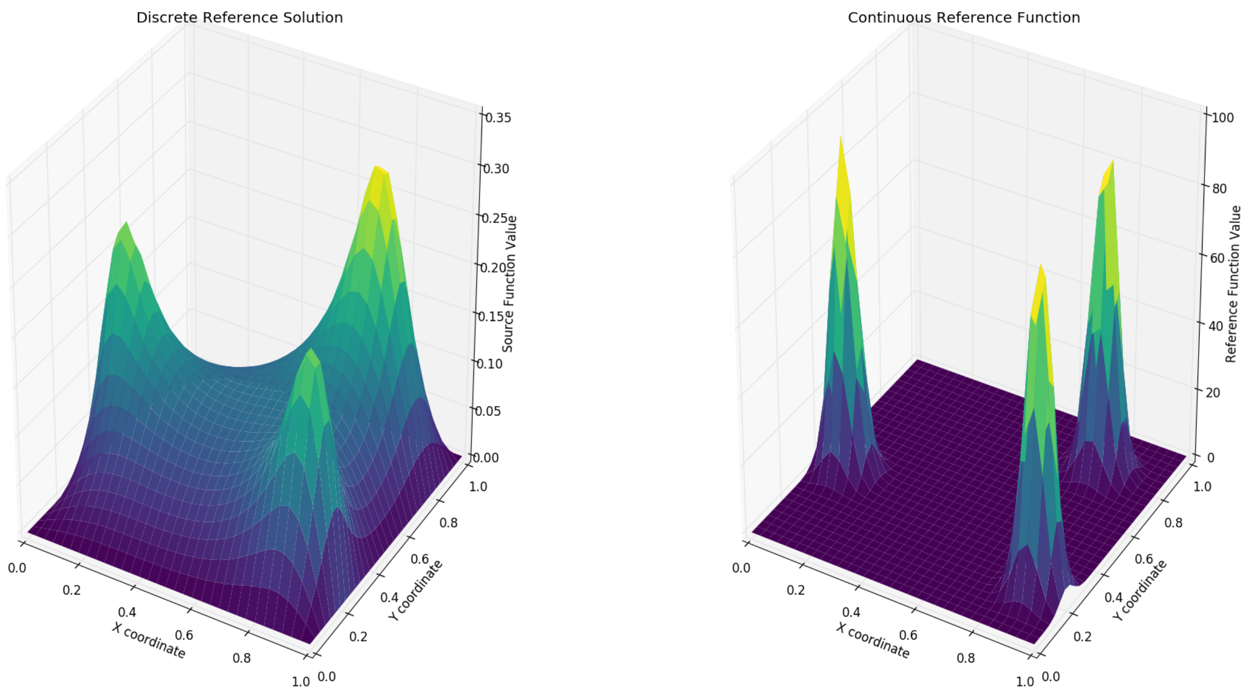


Figure 4.1: The discrete solution to the PDE $\bar{\boldsymbol{u}}^d$ (left) and the reference function $\bar{f}$ (right)

4.2 Gaussian Hyperparameters

The Source Inversion problem requires two hyperparameters, a and σ, to construct the Gaussian reference and source functions. While I ensured that the reference and source functions always used the same a and σ, it is important to note that changing the σ parameter has important ramifications for the problem, while changing a has little meaningful effect due to the scale invariance of $\boldsymbol{w}$. Due to this scale invariance, I set $a = 100$ for all configurations and tests.

Intuitively, the variance of each Gaussian source function represents the spread of its density across the discretized domain Θ. With a high variance, the source function is flatter (in a one dimensional setting, we would say that the Gaussian function has heavy tails), while if the variance is low, the density function is sharply peaked. When creating a field of such source functions, a high variance causes these functions to blend together, while a low variance results in distinct peaks at each k, l source function centroid, which rapidly diminish to zero. As a measurement metric for variance, I used the value of a given source function at its neighboring source function centroids. Pictorally, this metric is measured as follows. If we consider source function f_{23} which is centered at the 2nd row and 3rd column of the n x n source function matrix embedded in the mesh, then $f_{23}(k, l) = 100$ at its center, when $k, l = 2, 3$. We can find the value of f_{23} by evaluating this source function at the coordinates of its immediate neighbors' centroids (any of the other source functions on the mesh in Figure 4.2), as any neighbor will return the same value.

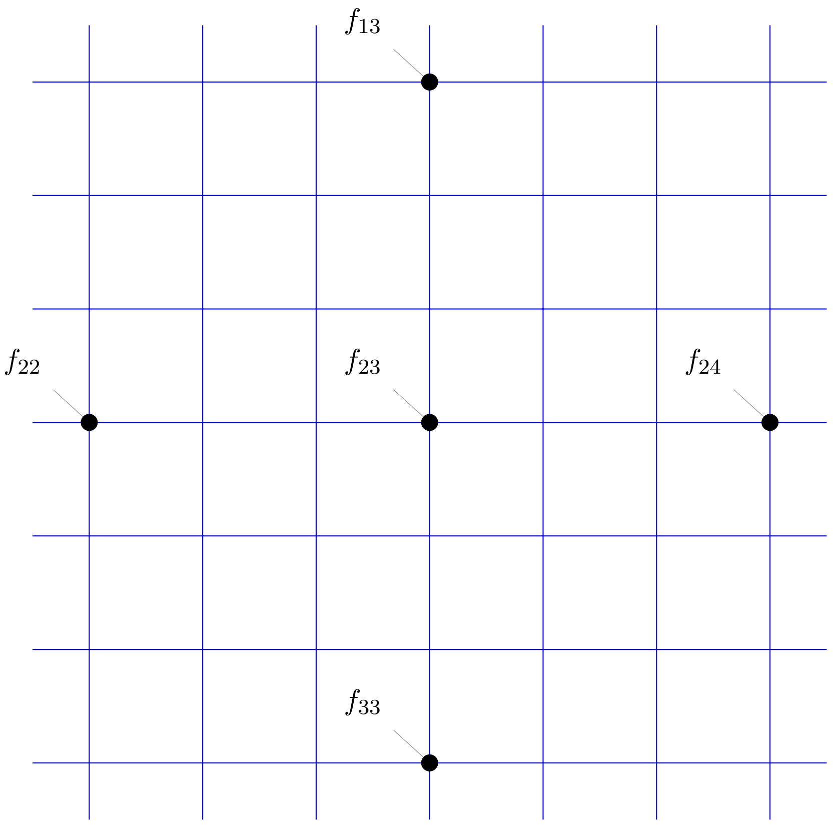


Figure 4.2: A small section of mesh with five source function centroids

Mathematically, we can formulate a simple problem to give a σ such that the value of a source function f at its immediate neighbors' centers is x% of the value at f's center (which, as aforementioned, is 100). As the mesh is created first in the problem, and the square submatrix positions for each source function directly thereafter, we note that the value of a source function centered at x_k, y_k evaluated at the center of its neighbor can be written as:

$$val = 100 \cdot exp\{\frac{-((x_{neighbor} - x_k)^2 + (y_{neighbor} - y_k)^2)}{\sigma}\} \tag{4.3}$$

Naturally, val will be a number less than 100 (as the value of the source function must be lower than its peak in all other locations, and so dividing out the 100 and taking logs we can set σ by choosing the desired value of a source function at the centroid of an immediate neighbor as a percentage of the source function's peak value.

We then can set σ as

$$\sigma = \frac{-((x_{neighbor} - x_k)^2 + (y_{neighbor} - y_k)^2)}{\log(pctval)} \tag{4.4}$$

It is also noted that due to the fact that the source functions are on a grid, exactly one of either the x or y coordinate of the source function and its neighbor will be identical, and so in practice the equation to set sigma (arbitrarily choosing the source function f_{23} and neighbor f_{24}), the equation simplifies to:

$$\sigma = \frac{-(y_4 - y_3)^2}{\log(pctval)} \tag{4.5}$$

Having found a way to choose σ, we can now see how the problem changes when the *pctval* metric is varied. Sager, Bock, and Reinelt solve a one dimensional version of the Source Inversion Problem [25] also using $a = 100$, and a σ such that *pctval* is 45%. One must note, however, that in the two-dimensional setting, a higher variance has more severe ramifications, as the value of the field (the sum of all source functions) at any given point does not receive contributions from the neighboring source functions in the only the x dimension, but the neighboring functions in the y dimension as well. Additionally, a higher variance results in the value of any point in the field being tangibly affected by not only its immideate neighbors, but also any nearby functions (which again, results in a greater effect in the two-dimensional scenario compared to the one-dimensional scenario). Below are examples of the field of source functions and the solution to the source PDE $\boldsymbol{u}^d$, given different values of σ. These results are garnered with $n^2 = 16$ source functions on a 32 x 32 mesh.

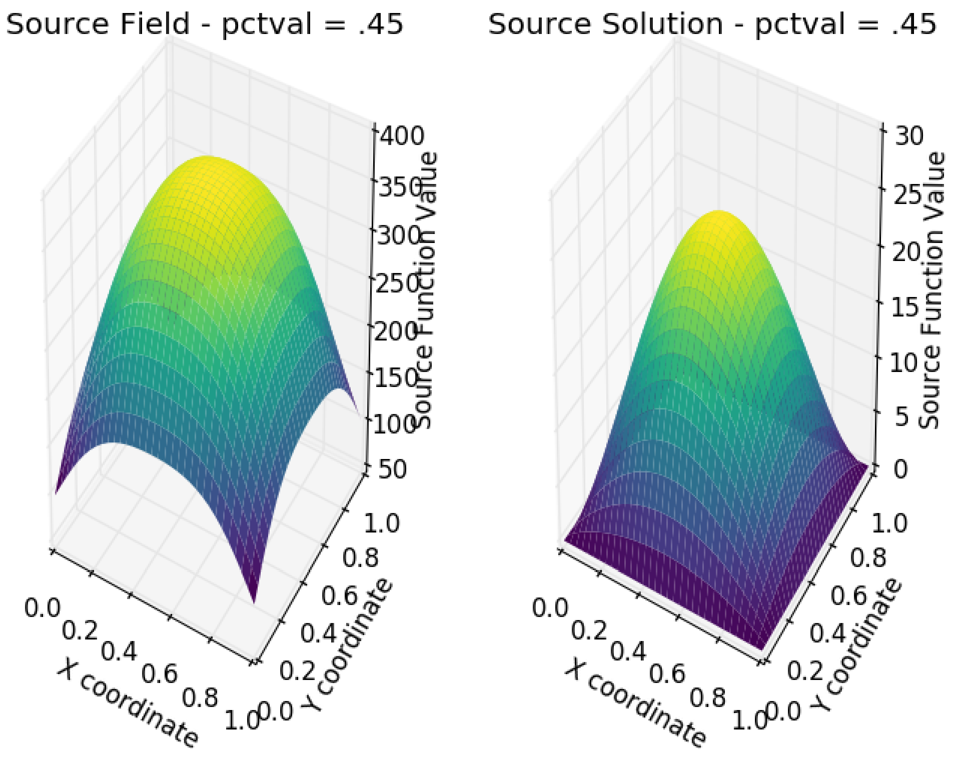


Figure 4.3: The source field and solution to the source PDE (*pctval* = .45, $n = 4$, $m = 32$)

As one can see, the high variance results in each source function bleeding into its neighbor's sphere of influence too much – the field is increasingly uniform in the center of the domain, and results are somewhat spurious. If we similarly visualize the reference function and the solution to the reference PDE in Figure 4.4, one notices that the individual Gaussian component centers are difficult to spot in the solution due to the high variance of each component of the function.

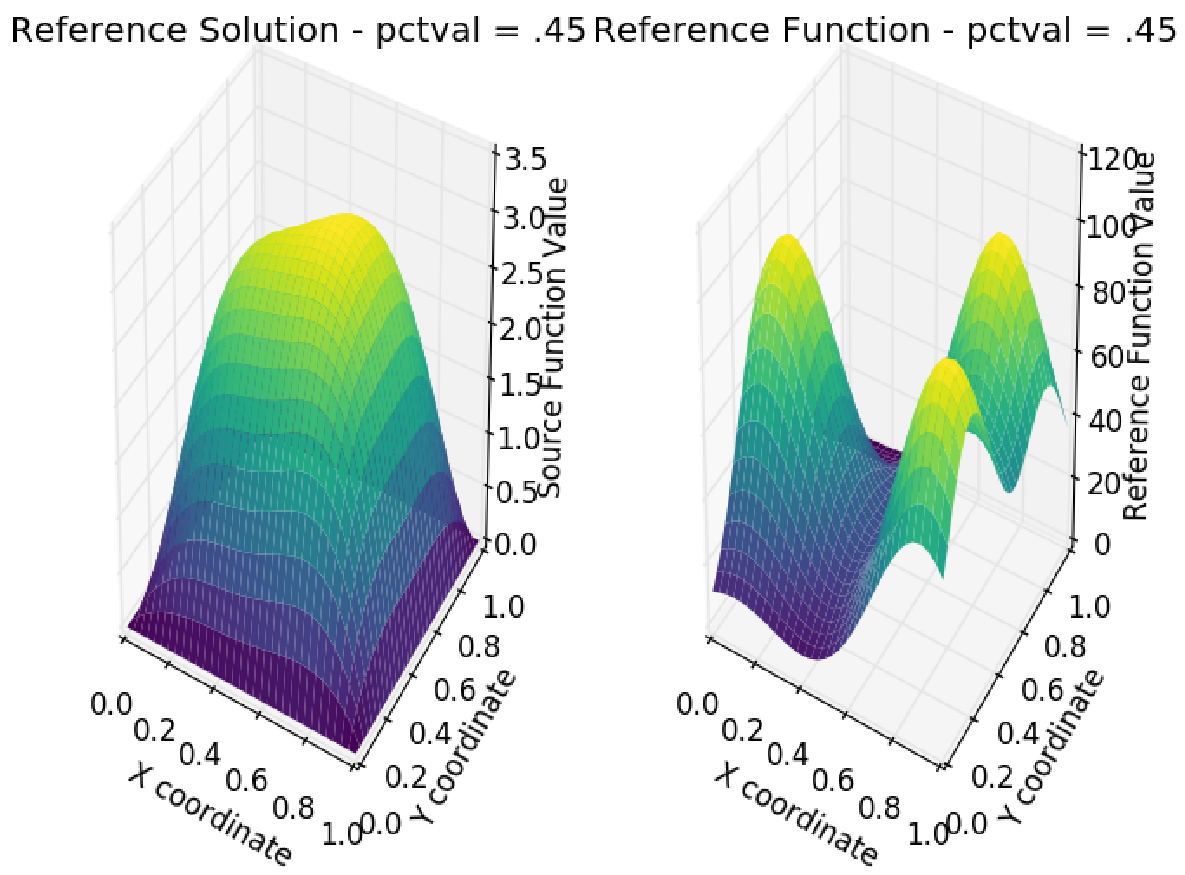


Figure 4.4: The reference function and solution to the ref. PDE ($pctval = .45$, $n = 4$, $m = 16$)

Mathematically, this overlap has serious consequences. When solving the optimization problem, high variance of each Gaussian component will cause many different parameter configurations to result in a near-optimal result (as the reference solution is not obviously composed of three individual functions but is rather a smeared-out mass in the center of the domain, as one can see in the left plot of Figure 4.4), and crucially, different integer solutions also yield nearly the same result, as it is difficult to distinguish between neighboring source functions. In essence, there is too much noise to properly distinguish between the source functions that should and should not be activated.

As the *pctval* shrinks, so too does the variance of each Gaussian component. As a result the source functions are more clearly distinguishable from each other in the field, and the reference solution becomes sharply peaked at the reference function's three Gaussian component centroids. As the noise disappears and the separation between elements of the source and reference field become sharper, so too does a clear continuous (and integer) solution become more obvious, as activating source functions has a more local and less smeared-out effect.

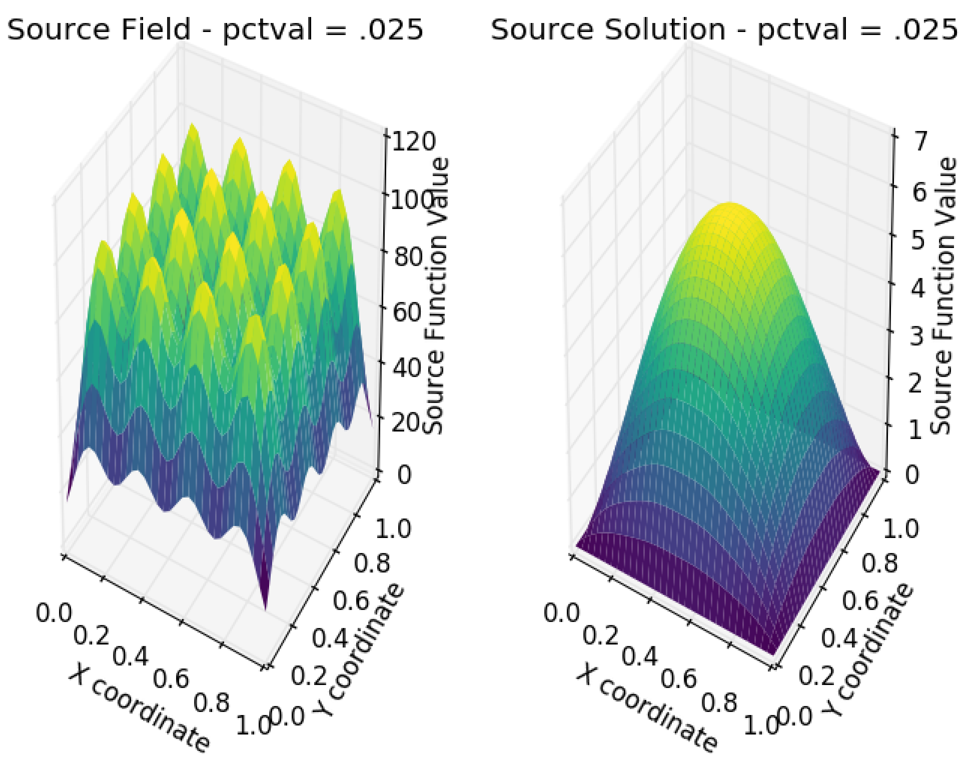


Figure 4.5: The source field and solution to the source PDE (*pctval* = .025, $n = 4$, $m = 32$)

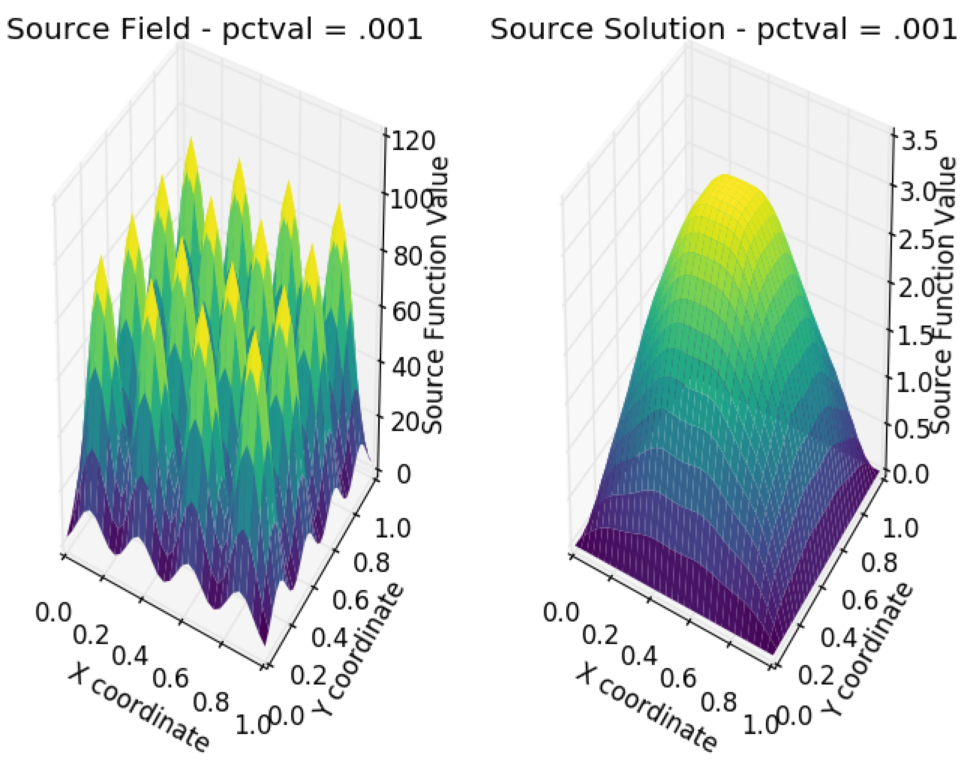


Figure 4.6: The source field and solution to the source PDE (*pctval* = .001, $n = 4$, $m = 32$)

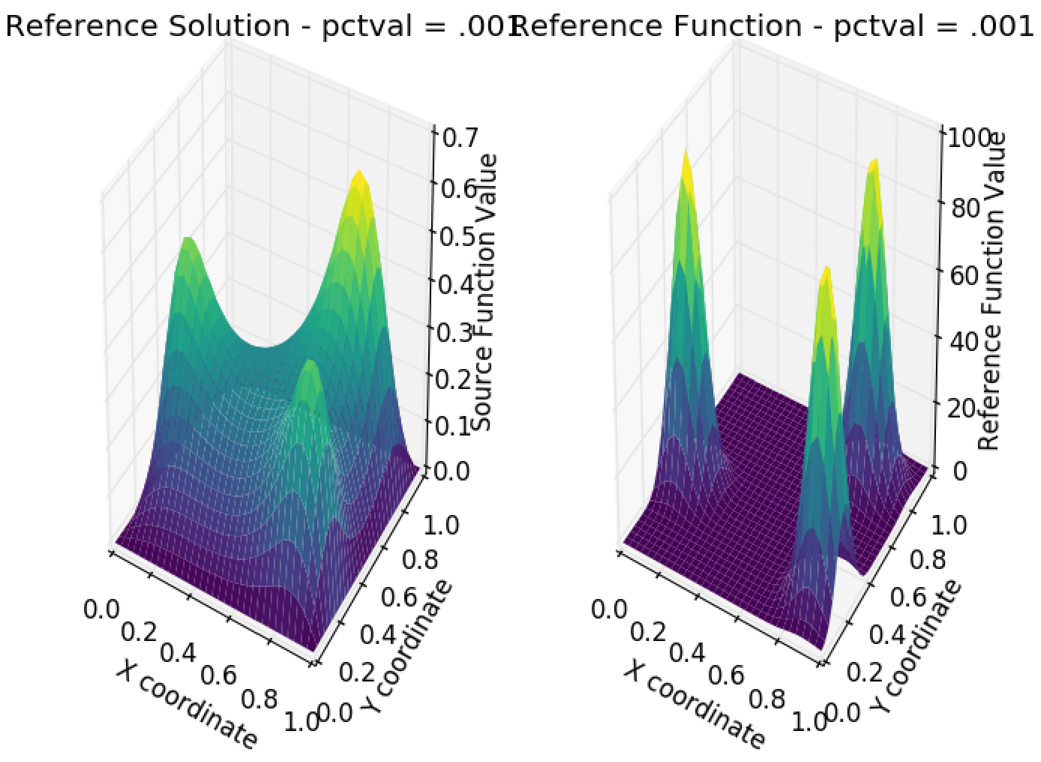


Figure 4.7: The reference function and solution to the ref. PDE ($pctval = .001$, $n = 4$, $m = 32$)

In choosing the hyperparameters of the Source Inversion problem, it is important to note the trade-off between variance and solving the optimization problem with integer constraints. By choosing a high variance, it is difficult to find an optimal solution that does not simply assign a low value of w_{kl} to each source function. While finding continuous and integer solutions may be difficult, the difference between the continuous and integer solution is assured to be significant, as the continuous solution will have many sources activated with low values of w_{kl}. Conversely, if the variance is tiny, then the continuous solution will end up being sparse to the extent that it is close to the integer solution, or rounding up the largest S optimal continuous w_{kl} values to 1, and rounding all others to zero will result in the optimal integer solution every time, thus rendering the use of any (computationally more expensive) integer algorithm pointless (i.e. one might as well solve the continuous problem and use a rounding heuristic to get the right answer). In an effort to balance these two issues, I used $pctval$ parameters of 1% and 2.5%, which ensured that each Gaussian function in the source field and the reference solution has a distinct peak, while maintaining enough variance so that the Gaussian components of the source and reference functions still overlap significantly. Under various mesh sizes and source function numbers, the value of the variance parameter σ is:

Value of σ	σ when $pctval = .01$				σ when $pctval = .025$			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	.0136	.0136	.0136	.0136	.017	.017	.017	.017
$n = 8$	.0034	.0034	.0034	.0034	.0042	.0042	.0042	.0034
$n = 16$	N/A	.0009	.0009	.0009	N/A	.0011	.0011	.0011

Figure 4.8: Values of the source function variance hyperparameter σ for different n, m

4.3 The Source Function Upper Bound Parameter (S)

The parameter S governs the number of source functions which may have nonzero w_{kl} coefficients. In the continuous case, as the source function is composed of three Gaussians, $\sum_{k,l} w_{kl} \sim 3$ in all cases. In the integer case, however, due to the inexact precision of turning sources on and off, it is possible that a combination of four (or more) source functions is the best approximator to the PDE in terms of minimizing overall reconstruction error. To reflect as much, I tested two values of S. First, I used $S = 3$, hypothesizing that exactly 3 source functions would be activated, each one being sufficiently close to one of the Gaussian components of the reference function. This proved to be the case; for completeness, I tested $S = 5$, assuming that the case could arise whereby source functions are not adequately close enough to each of the reference Gaussian bumps, and so some of the source functions offset each other in order to minimize the error – in practice, this did not occur unless the integrality constraints were already violated. Consequently, all results presented in this section assume that $S = 3$.

4.4 Optimizing the Penalty Term

One of the most computationally demanding tasks was to find the optimal hyperparameter α^* of the penalty function algorithm, as I was forced to selectively search using algorithm 9. Unfortunately, discerning a reasonable error metric for α is difficult, as algorithm 9 attempts to find the best α to solve the integer problem (i.e. minimizing reconstruction error of the source solution using the optimal $\boldsymbol{w}^*$ *subject to* each element in this optimal vector being binary). Consequently, a classic error metric such as solving the penalty algorithm a number of times and measuring reconstruction error for each α is not sensible, as α values which return a lower reconstruction error with continuous values of $\boldsymbol{w}^*$ *are not* preferred to α values which encourage integer $\boldsymbol{w}^*$, even if returning a higher construction error.

Under each configuration of n and m, I implemented algorithm 9 in order to find the α which returned an integer solution. The optimal (or best found) values of α^* are below:

Value of α^*	α^* when *pctval* = .01				α^* when *pctval* = .025			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	.01625	.017	.017	.018	.04	.042	.044	.019
$n = 8$	.000125	.00015	.000175	.019	.0005	.00055	.0006	.019
$n = 16$	N/A	N/A	N/A	.019	N/A	N/A	N/A	.019

Figure 4.9: Values of penalty function hyperparameter α^* for different n, m when $S = 3$

4.5 Tested Algorithms

In order to perform time and accuracy benchmarking, I tested the following five gradient-based algorithms (in a later section, I contrast the results of each gradient-based algorithm to non-gradient-based MINLP branch and bound baseline).

Algorithm	Description
Continuous (sum control)	The algorithm which uses the sum of source functions as the control (scales well in n but poorly in m) and finds optimal values of $0 \leq w_{kl} \leq 1$
Continuous (w control)	The algorithm which uses each w_{kl} as the control (scales poorly in n but well in m) and finds optimal values of $0 \leq w_{kl} \leq 1$
Penalty	The algorithm which adds $\alpha \sum_{k,l} \lvert w_{kl}(1-w_{kl}) \rvert$ in order to encourage integer solutions
Two-Step	The algorithm which takes the output of the continuous algorithm (w control) as the $\boldsymbol{w}_{init}$ input to the penalty algorithm
MINLP	A branch and bound algorithm which finds integer solutions directly using a divide-and-conquer global search

4.5.1 Parameter Configurations

Given the above algorithms, I tested the following configurations of the Source Inversion problem in order to find scalability limits and to compare algorithm performance. All tests were conducted on a laptop with a Intel Core i7-7500U CPU @ 2.70GHz with 16gb of RAM. Note that none of these operations were run in parallel. I decided to use $n = 4$, $n = 8$, and $n = 16$ on each mesh in order to track the increasing computational complexity due to increasing the number of source functions. The following table outlines the configurations of parameters tested. As reported in the following sections, I calculated time complexity and error for each configuration, using the L_1 error metric (in practice this varied over n but *not* over m). Also note, not pictured below, that this entire table was tested twice: once for Gaussian source functions with σ such that $pctval = .01$ and once when $pctval = .025$. In the following section, $pctval = .01$ is assumed, as changing $pctval$ to .025 resulting in minor temporal differences across algorithms results (a slight decrease in time complexity), but not structural differences. It is interesting to note that while the time trends do not change as σ changes, increasing σ, that is, introducing a higher variance, uniformly reduces the time of all tested configurations.

Configurations Tested	$pctval = .01$				$pctval = .025$			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	✓	✓	✓	✓	✓	✓	✓	✓
$n = 8$	✓	✓	✓	✓	✓	✓	✓	✓
$n = 16$	X	✓	✓	✓	X	✓	✓	✓

Figure 4.10: Parameter configurations tested for the Source Inversion problem

4.5.2 Optimization Hyperparameters

SciPy implementations of all optimization algorithms were used. In the continuous (w control), heuristic integer, penalty, and two-step algorithms, optimizing with w_{kl}'s as the controls was done using sequential quadratic programming (SQP) using SciPy's SLSQP. The convergence criteria for the objective functional was $1e-5$ when $n = 4$, and $1e-10$ when $n = 8$, 16, with a maximum of 400 iterations. In practice, SLSQP converged in a few hundred iterations for the continuous (w control)/heuristic integer algorithms for any n, and for the penalty/two-step algorithms when $n = 4$. Above $n = 4$, the penalty/two-step methods did not converge; the objective function value was still falling very slowly at 400 iterations. In the two-step algorithm, the second stage of the optimization problem was run using SLSQP with the same functional tolerance as the other algorithms. This converged each time in less than 20 iterations, likely due to the convexity of the problem. For the sum control algorithm, I used SciPy's implementation of BFGS, which was capped at 150 iterations and had a functional tolerance of $1e-7$ for $n = 4$ and $1e-10$ for $n = 8$, 16.

4.6 Results and Discussion

The following results were obtained after testing each algorithm with the aforementioned parameter configurations. Note the following abbreviations and units:

1. **Time is measured in seconds**
2. **TO denotes 'timeout'**
3. **N/A means that there is no error information due to algorithm timeout**
4. **X means that the configuration was not tested**
5. *** means that the two-step/penalty algorithm violated its integer constraints**

As the continuous algorithm with w as a control for $n = 16$ took over four days to run, the integer algorithms were not tested for $n = 16$. Though they likely would have converged with properly chosen parameters, choosing a proper α that needs precision to six or more significant figures proved quite difficult. In other words, the penalty method was generally not robust.

Cont. (w control)	Time Complexity				Error			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	68.23	65.76	142.01	190.11	.028	.043	.031	.031
$n = 8$	1657	1762	2753.49	4565.04	.0096	.0096	.0095	.0095
$n = 16$	X	5126.86	63,034.54	TO	X	.0019	.002	N/A

Figure 4.11: Results for the continuous (w control) algorithm

Cont. (sum control)	Time Complexity				Error			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	343	657	3110	TO	.0367	.042	.057	N/A
$n = 8$	638	1051	7022	TO	.045	.0407	.065	N/A
$n = 16$	X	13720	TO	TO	X	.019	3	N/A

Figure 4.12: Results for the continuous (sum control) algorithm

Heuristic Integer	Time Complexity				Error			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	68.23	65.76	142.01	190.11	.076	.071	.071	.071
$n = 8$	1657	1762	2753.49	4565.04	.01998	.0209	.021	
$n = 16$	X	5126.86	63,034.54	TO	X	.019	.02	N/A

Figure 4.13: Results for the heuristic integer algorithm

Penalty	Time Complexity				Error			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	105.86	114.83	171.63	235.84	.077	.071	.071	.071
$n = 8$	2402.21	2760.45	3324.3	4839.23	.0077*	.0077*	.0077*	.0077*
$n = 16$	X	TO	TO	X	N/A	N/A	N/A	N/A

Figure 4.14: Results for the penalty function algorithm

Two Step	Two Step - Time Complexity				Two Step - Error			
	m=16	m=32	m=64	m=128	m=16	m=32	m=64	m=128
$n = 4$	126.23	150.26	237.96	291.43	.077	.071	.071	.071
$n = 8$	4321.19	4456.21	5134.54	9345.24	.0062*	.0062*	.0062*	.0062*
$n = 16$	X	TO	TO	TO	X	N/A	N/A	N/A

Figure 4.15: Results for the two-step algorithm

4.6.1 Time Complexity

Glancing across the tables, it is easy to confirm that algorithm runtime generally increases in the number of source functions used. Interestingly, however, it appears that for the continuous (sum control) algorithm, the mesh parameter m dominates the n parameter in terms of complexity. That is, the runtime of the algorithm scales well in n, but poorly in m. Using $n = 4$, the continuous (w control), heuristic integer, penalty function, and two-step algorithms all had $n^2 = 16$ control parameters, one for each w_{kl}. The sum control algorithm, as always, had one control. Yet the bulk of the computational work for the latter four algorithms came from the controls – thus as m was increased, the runtime of the algorithms increased by only moderately, while the sum control algorithm run time increased exponentially, consistently timing out when $m = 128$. See Figure 4.16 for a graphical representation.

The strength of the sum-control algorithm is seen when n is increased. At $n = 8$, the sum-control algorithm still has a single control, while the other algorithms have 64 controls. Here, the sum-control algorithm marginally outperforms all of the other algorithms in terms of time complexity when m is small, though the difference is not compelling enough to make up for the sum control algorithm's poor accuracy. At $n = 16$, however, the sum control algorithm's runtime is already high for a small m, and quickly times out as m increases. Additionally, the sum control algorithm is generally inaccurate, even with its continuous solutions. Without any precise delineation, it is clear that the four algorithms which use each w_{kl} as a control scale well in mesh size m, but poorly in source function number n^2, while the continuous (sum control) algorithm does the opposite, scaling fairly well in source function number n^2, and poorly in mesh size m. While it is possible that the continuous (sum control) algorithm's accuracy woes could be solved by increasing the tolerance in the first stage (when BFGS is used), any increase in accuracy would come at the expense of an increase in the already oft-untenable time complexity.

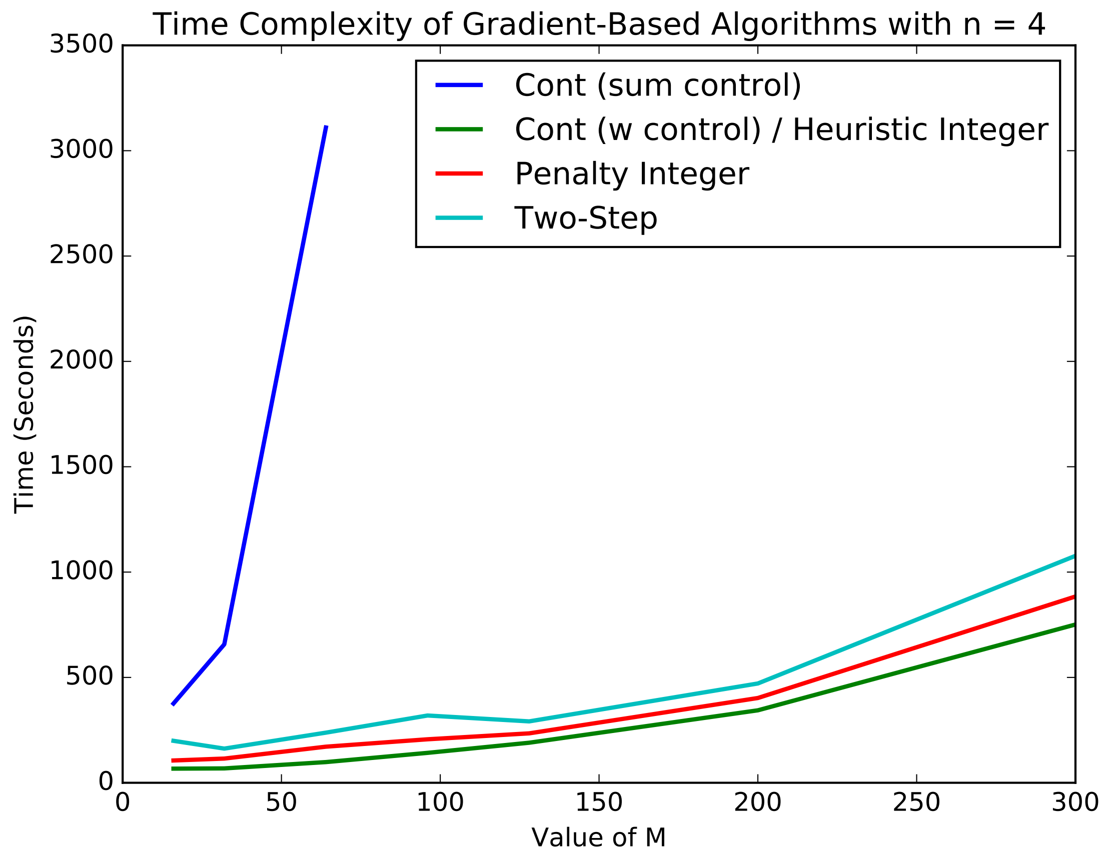


Figure 4.16: Time complexity across mesh size m of all algorithms when $n = 4$

4.6.2 Accuracy

The comparison of time complexity for the above parameter configurations is pointless, however, without a comparing the accuracy of the solutions as well. Naturally, we desire to trade off time complexity and accuracy to our liking, in that we tolerate a solution which takes a long time to find if we highly prize accuracy, or we prefer a fast solution if we want a rough estimate of optimality and a fast runtime. Before delving into a comparison of accuracy between the algorithms, it is imperative to note a few structural differences between the algorithms.

Continuous and Integer Solutions

For a problem such as this one, the continuous solution is the relaxation of an integer solution, That is, the requirement of integrality imposes more constraints on the Source Inversion problem. Consequently, the solution to the continuous relaxation (i.e. $0 \leq w_{kl} \leq 1$) of the Source Inversion problem is guaranteed to have an error equal to or lower than that of Source Inversion problem with binary w_{kl}'s.

Penalty and Two-Step Algorithms

Both of these algorithms proceed by adding a penalty term – which relies on hyperparameter α – to the objective functional which encourages integer solutions. In this problem, the objective functional was extremely sensitive to α for penalizing non-binary w_{kl} values with both $L1$ and $L2$ norms. While α was relatively easy to find for the $n = 4$ case, when the number of source functions increased, α became increasingly small, and the slightest of perturbations from the optimal value yielded non-integer values – i.e. the penalty and two-step methods were not robust. Consequently, the penalty and two-step algorithms returned very good error values (quite close to the continuous error) in the $n = 8$ and $n = 16$ cases, but violated the integer constraints and produced non-binary optimal w_{kl} values. In practice these algorithms did converge to optimal w_{kl} values that were closer to being binary than the output of the continuous algorithm (for instance, two w_{kl}'s were 1, two were between 0 and 1, and all other values were zero), but I could not find perfect binary solutions, even when testing α by increments of $1e - 7$. In the $n = 4$ case, however, integrality constraints were satisfied and the heuristic integer, penalty function, and two-step algorithms are returned the same exact solutions.

The Continuous (Sum Control) Algorithm

While the other four algorithms solved for the optimal w_{kl}'s in one go, the continuous (sum control) algorithm had two steps. As mentioned in the previous chapter, the second step of the continuous (sum control) algorithm is a convex optimization problem, *provided optimality of the first step.* That is, when the sum of source functions at each mesh vertex is optimized, the resulting vector must be a global minimum of the first stage problem in order for global optimality of the w_{kl} values to be assured in the second stage. Unfortunately, there is not certificate of global optimality for the first stage. Additionally, BFGS did not always terminate due to convergence criteria, but sometimes due to an iteration maximum. While I chose the maximum to be 150 iterations, and a tolerance of $1e - 10$, more work would need to be done to actually find the optimal iteration number and tolerance (I did a bit of testing but could not afford to focus entirely on it). Consequently, I would venture a guess that the higher error rates experienced by the continuous (sum control) algorithm are not a result of flawed ideology, but rather imperfect convergence and iterations limits. Importantly, however, changing such limits would affect time complexity, and without additional information, it is not possible to quantize the trade-off.

General Accuracy Trends

As aforementioned, the accuracy of the continuous relaxation (i.e continuous (w control)) is always the lower bound to the integer solutions. Indeed, we find this to be true. The accuracy of the heuristic integer solution is as good or better than the penalty and two-step algorithms in every case, and is much more time-efficient than the two-step algorithm. While the continuous (sum control) algorithm has desirable properties for scaling and time complexity, it is noted that it is quite inaccurate – even more inaccurate than the integer algorithms in the $n = 8$ case; this could demonstrate improper convergence criteria (too large a tolerance), which, if properly implemented, could increase the runtime of the continuous (sum control) algorithm quite a bit. It is important to note that in Figure 4.18, while the penalty and two-step algorithms appear to have better accuracy than the heuristic integer algorithm, the former two algorithms violate their integrality constraints at $n = 8$ (but not at $n = 4$), and are thus invalid. Additionally, the penalty and two-step algorithms timed out at $n = 16$, and thus do not have any error values.

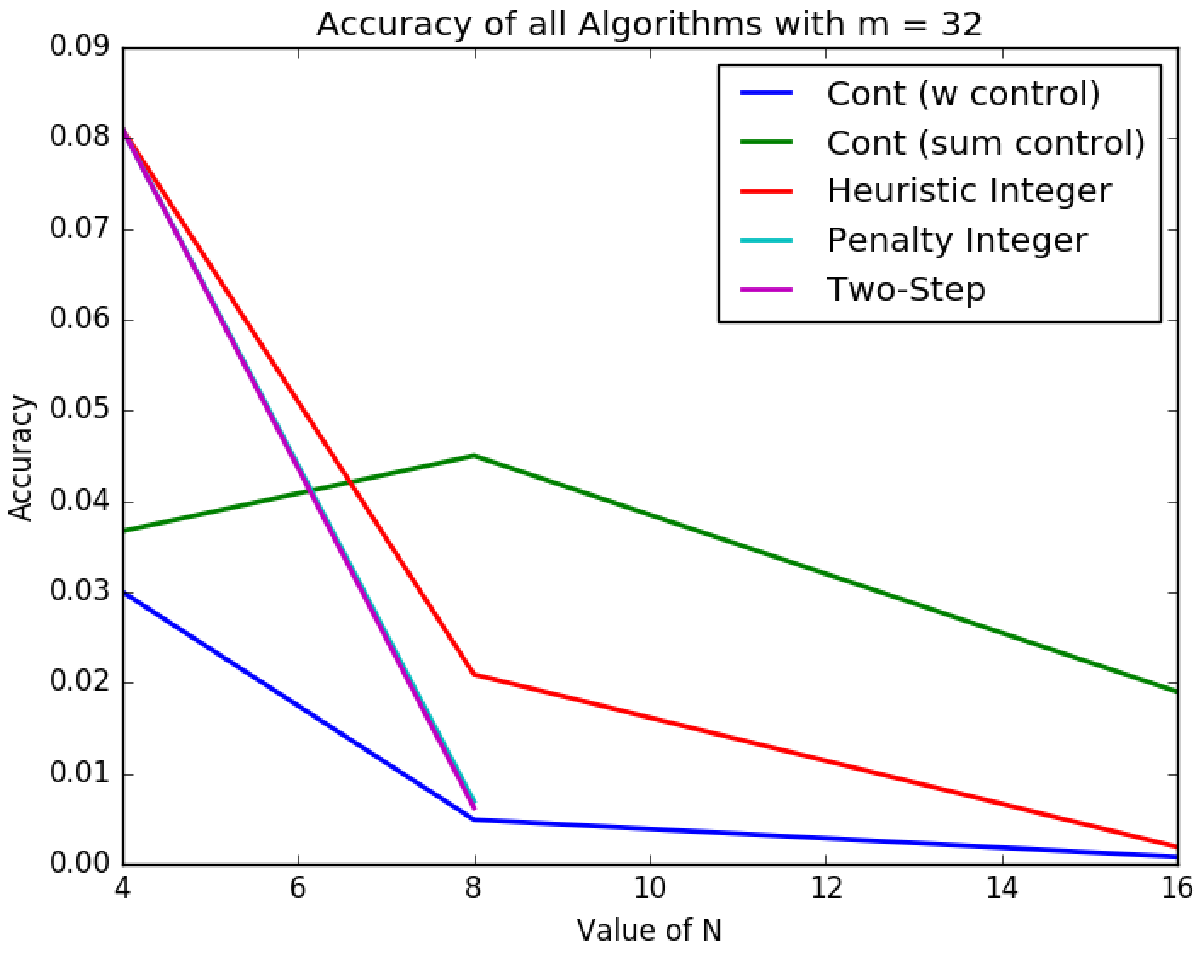
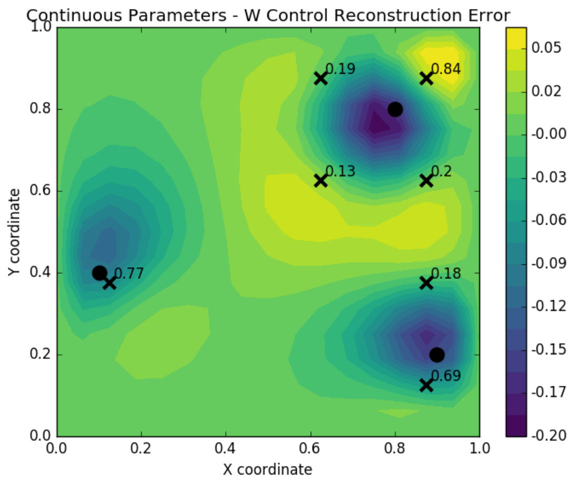


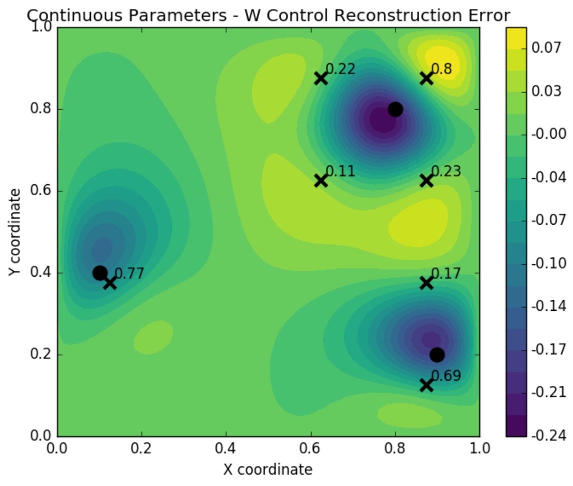
Figure 4.17: Accuracy of all algorithms across values of n

4.7 Error Contour Plots

As a more intuitive way of exploring error, the following contour plots present an error gradient over the entire discrete domain Θ. It is clear that the error changes little when the mesh is refined – the number of source functions and their general locations are still the same, and thus the same source functions are usually activated as integers (or in the continuous case, the magnitudes of w_{kl}'s does not change much). For example, if we compare the $n = 4$ and $m = 16$ case and the $n = 4$, $m = 64$ case when using the continuous (w control) algorithm, the nonzero w_{kl}'s are *exactly* the same. The magnitudes are perturbed only slightly (.11 $\rightarrow$ .14, or .20 $\rightarrow$ .23), and the error rate changes only a bit. The distribution of error, however, is also largely the same. This is depicted in Figure 4.19 below. Note that each black 'x' corresponds to an activated source function, and the number above and to the right of the 'x' is the value of its w_{kl} coefficient. The black circles correspond to the centroids of the Gaussian components of the reference function.



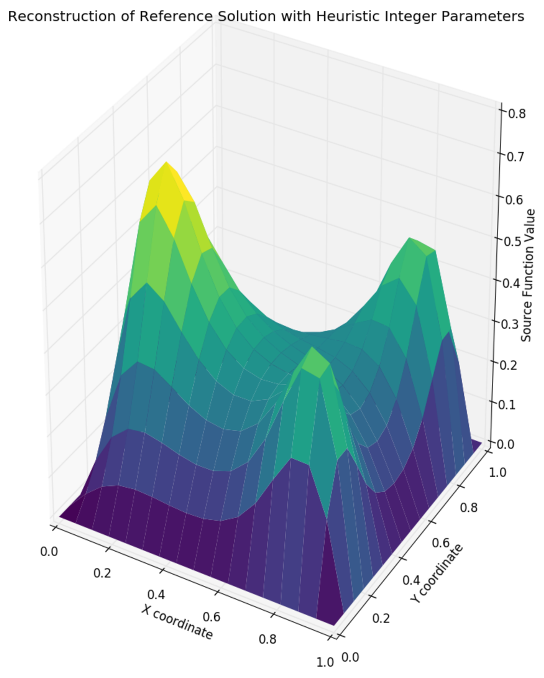
(a) Error when $n = 4$, $m = 16$



(b) Error when $n = 4$, $m = 64$

Figure 4.18: The variation in error over mesh size

Correspondingly, the reconstructed solution to the reference PDE does not vary much in terms of error across mesh sizes. One must note, however, that as both the reference and source PDE are discretized in a certain mesh (i.e. 16 x 16 or 64 x 64), then the reconstruction and reference solutions will look quite different across mesh sizes, despite the activated source functions being nearly the same. For instance, taking the heuristic integer algorithm this time, with $n = 4$, and $m = 16$, and $n = 4$ and $m = 64$:



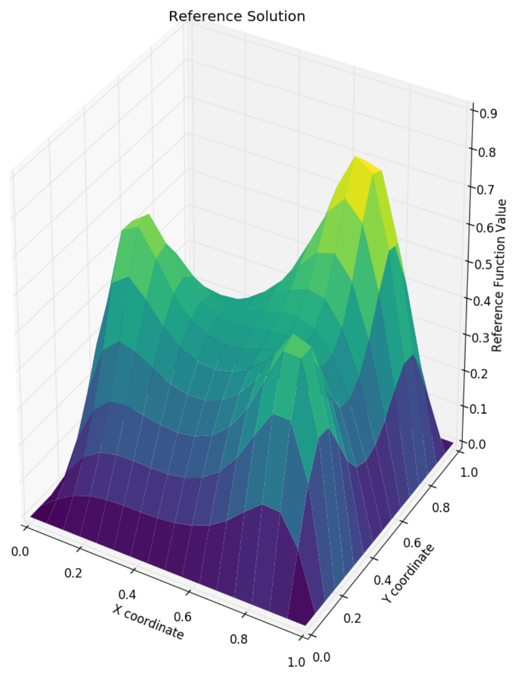


Figure 4.19: Reconstruction solution (left) and reference solution (right) ($n = 4$, $m = 16$)

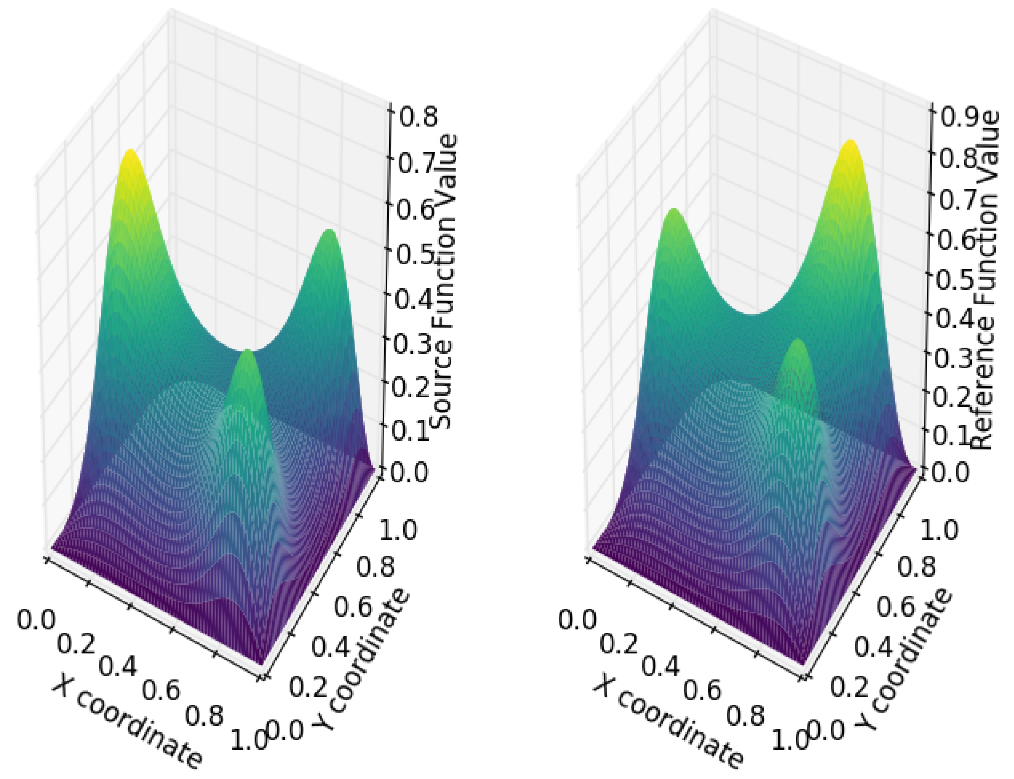
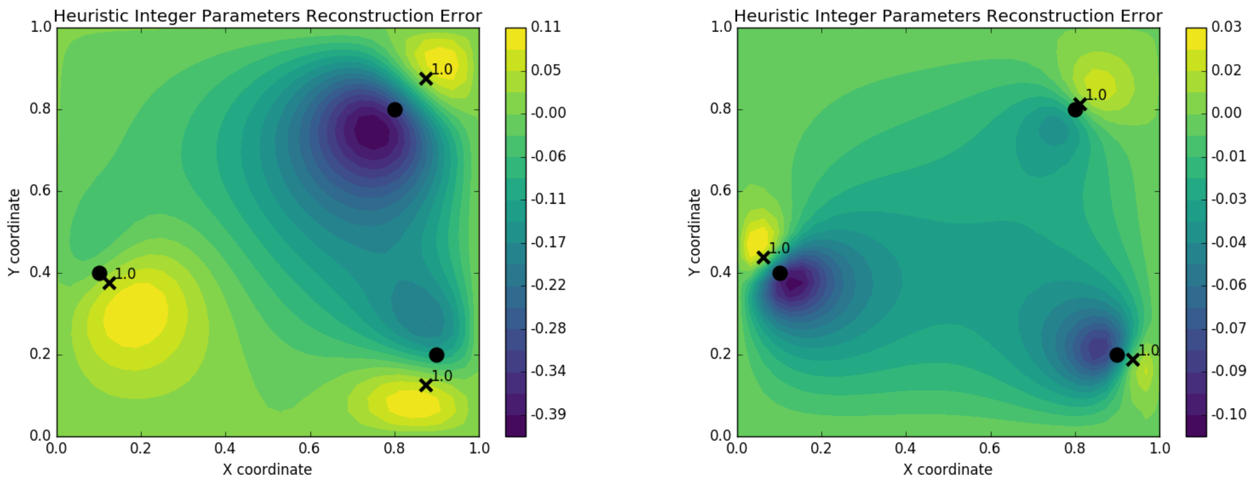


Figure 4.20: Reconstruction solution (left) and reference solution (right) ($n = 4$, $m = 64$)

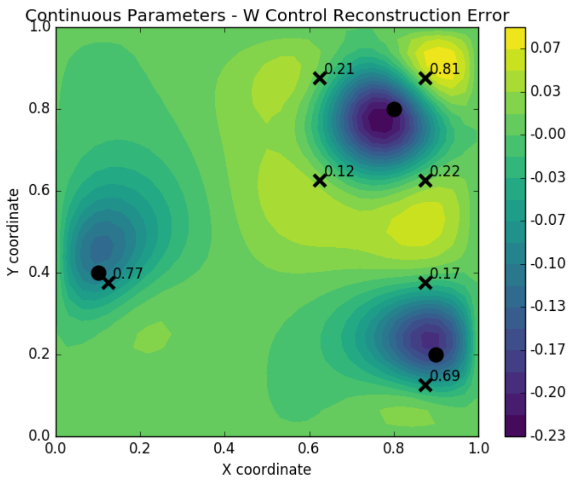
The above figures have demonstrated how accuracy is largely independent of mesh fineness in the sense that changing mesh size does not affect the source functions activated to create the PDE. If we increase the number of source functions, however, the case is quite different. Consider the optimal binary vector when $n = 4$ and $n = 8$, using a mesh size $m = 32$. The heuristic integer and continuous (w control) algorithms are presented.



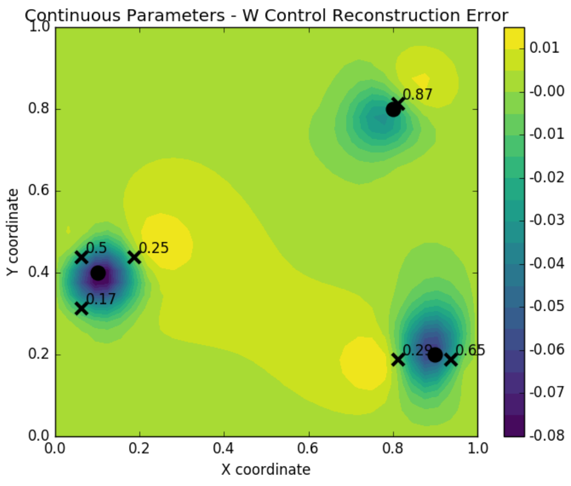
(a) Error when $n = 4$, $m = 32$ **(b)** Error when $n = 8$, $m = 32$

Figure 4.21: The variation in error (heuristic integer) over source function number n

As one can see, increasing the number of source functions dramatically decreases error, as



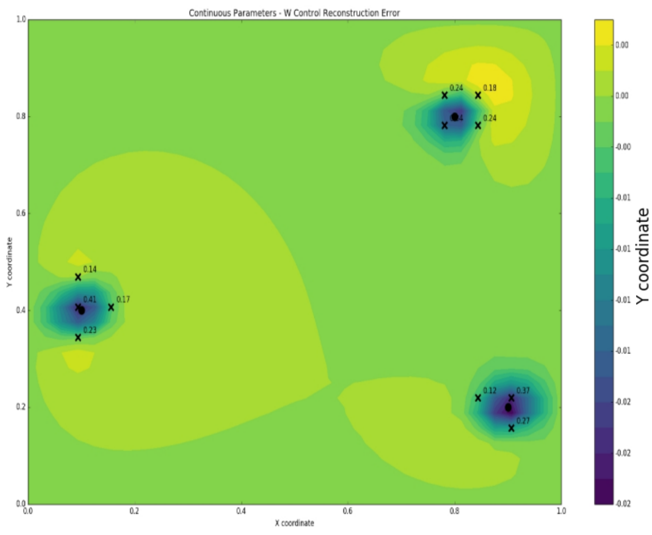
(a) Error when $n = 4$, $m = 32$



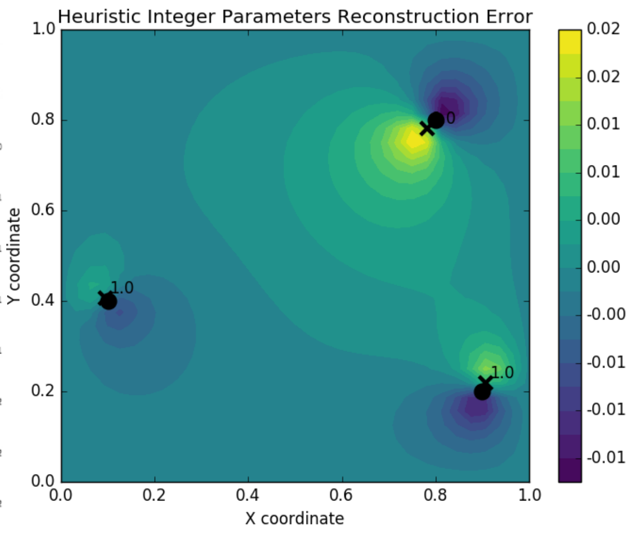
(b) Error when $n = 8$, $m = 32$

Figure 4.22: The variation in error (continuous (w control)) over source function number n

the increased number of source functions ensures that some source functions are closer to the centroids of the Gaussian components of the reference function as compared to situations with a smaller n. As n increases, the integer algorithms activate the source functions which are closer to the Gaussian centroids, as seen in Figure 4.22. As we continue to increase n, one might inquire about the asymptotic behavior of the optimal $\boldsymbol{w}$ vector – when $n \to \infty$, does the continuous (w control) algorithm collapse to three integer w_{kl}'s which correspond to source functions centered nearly on top of the three Gaussian reference centroids? If so, this asymptotic behavior would suggest that the continuous (w control) algorithm naturally tends towards integer solutions with a large enough n, possibly allowing us to drop the binary constraints. Unfortunately, while I cannot be certain of truly asymptotic activity, even at 256 source functions, the continuous solution does *not* collapse to integer points, though the heuristic integer solutions can be seen creeping closer and closer to the reference Gaussian centroids.



(a) Error when $n = 16$, $m = 32$



(b) Error when $n = 16$, $m = 32$

Figure 4.23: Continuous error (left) does not collapse to an integer solution at $n = 16$

4.8 Comparing Branch and Bound & Gradient-Based Algorithms

The algorithms presented so far offer a fairly comprehensive overview of how gradient-based optimization techniques may be applied to the Source Inversion problem. Although these techniques are assured to yield inexact results, their utility lies in the promise that their algorithmic efficiency makes up for any potential loss of accuracy. What remains, then, is a comparison of these four algorithms to an exact integer solution found using MINLP techniques. As described in Chapter 2, I compared the gradient-based algorithms to the MINLP algorithm branch and bound (implemented using the open source numeric optimization library CasADi, with IPOPT as the subproblem solver). Branch and bound proved to be extremely fast *and* accurate in most scenarios, and for most situations, seems objectively better than any gradient-based approach on all metrics. Yet the key difference between gradient-based algorithms and branch and bound lies in their different scaling paradigms. As aforementioned, the gradient based algorithms which create a control for each source function scale well in mesh size, yet poorly in the number of source functions. The gradient-based algorithm which uses the sum of source functions as a control scales poorly in mesh size, yet well in the number of source functions, though this algorithm was less accurate than all other algorithms used. The branch and bound algorithm fits neither of these paradigms, however, as it scales in the *total* number of variables. That is, branch and bound performs well when $n^2 + m^2$ is small, but performs poorly as the total number of variables is increased (whether by increasing the number of source functions or by refining the mesh). Consequently, while branch and bound outperforms all of the other algorithms in terms of time complexity in most of the scenarios – and as it is an integer programming algorithm, it returns the exact optimal integer solution – this third scaling paradigm causes branch and bound to perform terribly when refining the mesh. For instance, when the mesh is refined from 16 x 16 to 128 x 128, the gradient-based algorithms which scale well in mesh size see moderately increased time complexity. These algorithms still only need to find derivative information for n^2 variables. Yet as m increases to 128, branch and bound must handle and integer program with $n^2 + m^2$ constraints during each iteration of the algorithm – by increasing m to 128, the number of constraints increases from $256 + n^2$ to $16384 + n^2$! In the Source Inversion problem, $n <<< m$ by construction, and thus mesh refinement will often be the bottleneck whilst scaling. Finally, it is interesting to note how the number of nodes processed by branch and bound increased in n: when $n = 4$, roughly 50 nodes were processed in the optimization routine, regardless of mesh size. When $n = 8$, roughly 200 nodes were processed while when $n = 16$, the number was around 500.

In terms of accuracy, the branch and bound method and the heuristic integer method picked the same exact Gaussians to activate, and thus produced the same amount of error. In short, neither method is outright superior to the other for these limited testing configurations. One must note that the slight discrepancies in error between the algorithms and parameter configurations are a result of an error tolerance – in some cases, the optimal 'integer' w_{kl} was equal to .998 or the like, rather than precisely 1.0. When taking error tolerance into account, however, the branch and bound and heuristic integer algorithms had identical accuracy, and only varied in algorithm runtime.

As one can see in a graphical depiction of gradient-based and branch and bound algorithm runtimes, it appears that the gradient-based algorithms and branch and bound both scale in exponential time, yet branch and bound has a far higher coefficient than gradient-based algorithms. In other words, if $K = n^2 + m^2$, and branch and bound is $\mathcal{O}(a^K)$ while the gradient based algorithms are $\mathcal{O}(b^K)$, then $a >>> b$.

Title	Branch and Bound - Time Complexity				Branch and Bound- Error			
	m=16	m=32	m=64	m=128	m=16	m = 32	m = 64	m=128
$n = 4$	1.3	21.84	126.88	2412	.077	.071	.071	.071
$n = 8$	24.39	126.61	535.53	32268.28	.020	.021	.021	.021
$n = 16$	X	908.03	3921.83	TO	X	.0037	.0035	N/A

Figure 4.24: Results for the branch and bound algorithm

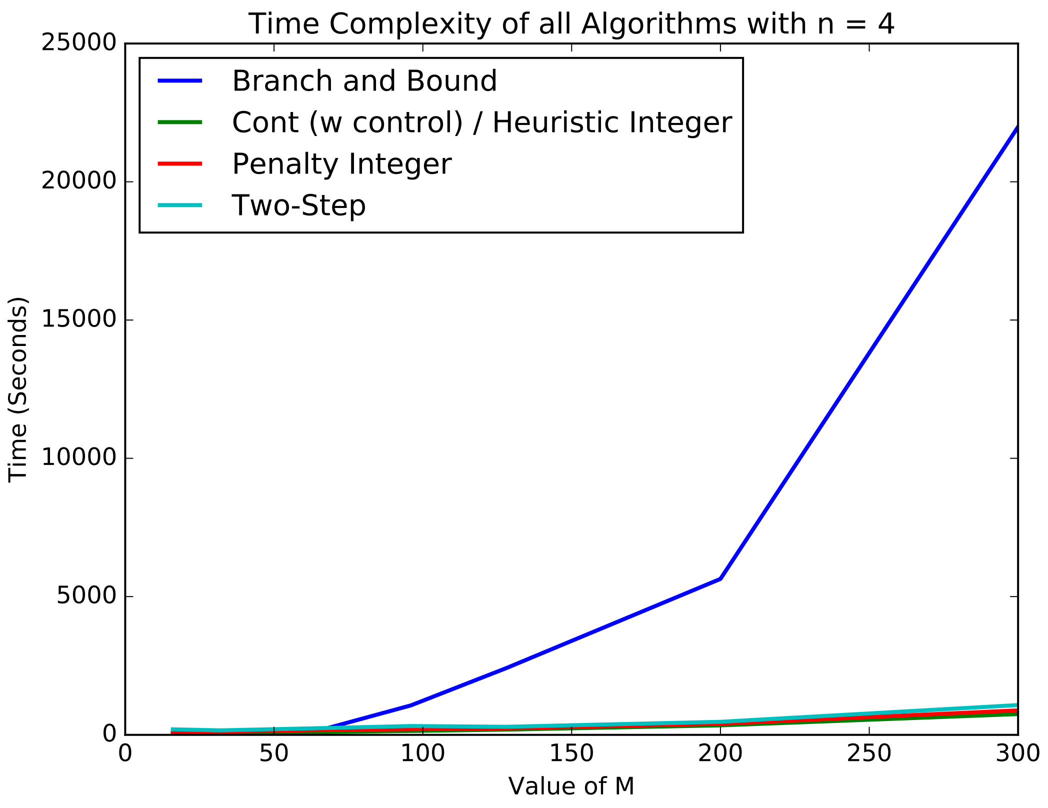


Figure 4.25: Gradient vs. MINLP complexity analysis across values of m

In order to better visualize the time complexities for low values of m, Figure 4.26 uses a log2 scale for time. In this figure, the exponential scaling of all algorithms is revealed by the linear trend (on the semi-log plot), at least at values of m greater than 128 (note that $m = 128, 200, 250, 300$ were tested), while the higher slope of branch and bound vis-a-vis the those of the gradient-based algorithms re-affirms branch and bound's higher scaling coefficient.

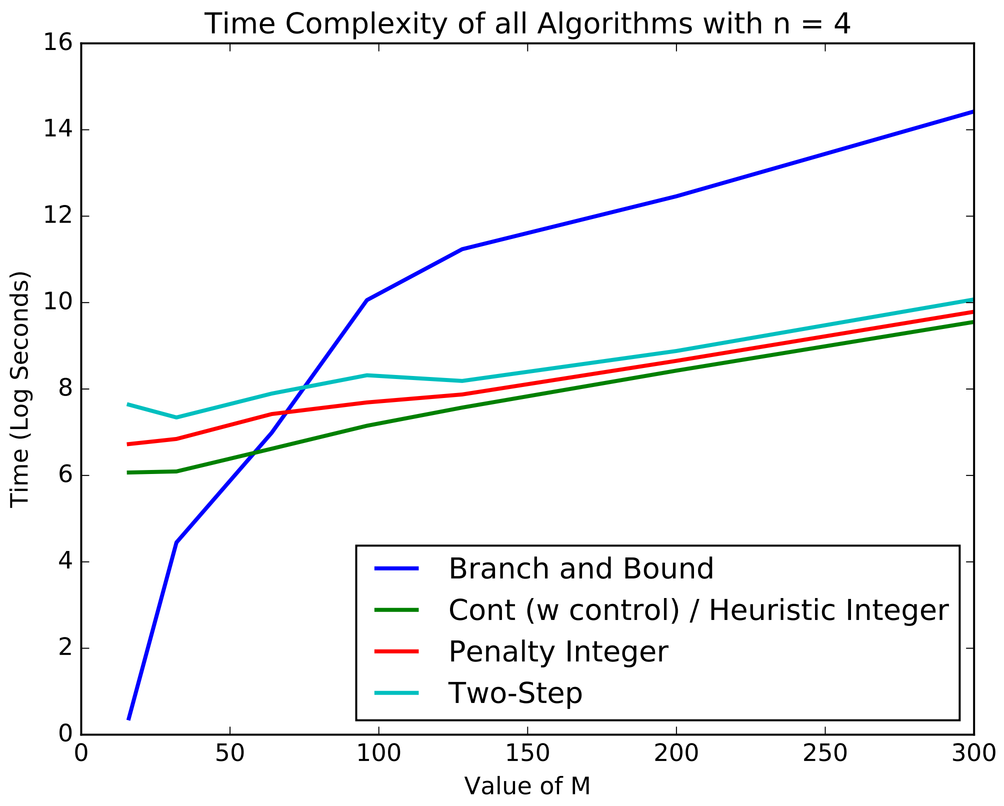


Figure 4.26: Gradient vs. MINLP complexity analysis across values of m (log2 scale)

Chapter 5

The Tidal Turbine Problem

5.1 Background Information

As the production of energy via renewable methods becomes increasingly socially and politically popular, advances in engineering methods are beginning to render tidal stream energy production economically viable as well. Yet tidal energy generation still requires high fixed installation costs and is a nascent concept with few large-scale industrial examples, though fields of tidal turbines are in construction/operation in Sihwa Lake, South Korea (254 MW, opened in 2011), La Rance, France (240 MW, opened in 1966), Swansea, Wales (400MW, in construction), and Pentland Firth, Scotland (398 MW, in construction) [28]. Note that these are the largest four tidal arrays in the world – three of them are either in construction or began operations within the past six years! As tidal energy production continues to gain steam – with increased scrutiny upon production efficiency driving costs down through R&D – optimization methodologies focusing on maximizing tidal stream energy generation as a function of turbine location are required.

Tidal turbines have several unique characteristics predisposing them to a well-defined optimization problem. For instance, compared to wind turbines, whose energy output depends on fickle wind speeds and direction, the movement of the tides is predictable [27]. Consequently, it is relatively easy to find regions which would be suited for tidal turbine arrays, though it has been noted a tidal range of at least 7m is required for cost-effective operation at the present level of technology [27]. The problem, then, is not to find the area where the turbines are suited, but rather to optimize the placement of the individual turbines within any suitable region (also known as micro-siting). Much of the difficulty in finding optimal placement results from the physical constraints of the problem: the local geography and bathymetry, the physics governing movements of tides and their interactions with each individual turbine (which is accurately modeled by a system of partial differential equations), and attributes of the water itself. Due to the complexity of the problem, there exists a broad trade-off between accuracy and computability. For instance, by simplifying the system of PDE constraints which governs the physics of the problem into a set of equations which are analytically tractable and less sophisticated, we are able to compute the optimal placement of turbines efficiently, at the expense of a less-realistic model (for instance, a MIP). Conversely, including the PDE constraints using the Navier-Stokes equation would result in an extremely accurate model, yet it would be nigh-impossible to solve. While I will enumerate upon previous MIP and PDE-based approaches and trade-offs, first I will flesh out the problem a bit more qualitatively. As a general note, the following optimization problems automatically incorporate the aforementioned adjoint approach to design, which is extremely effective in reducing computational complexity of PDE-constrained optimization problems without heavily affecting the accuracy of the resulting solutions. For an in-depth mathematical explanation of adjoint-based first-order optimization, see the Appendix.

5.2 Four Methods of Tidal Power Generation

Currently, there are four paradigms for tidal energy generation. The first method utilizes a dam, called a tidal barrage, which has sluice gates that allow water to flow into a bay during high tide (the dam is built across the entrance of the tidal inlet). Tidal turbines are a fixture of these dams so that water flows through the turbines into the bay during high tide, and back through the turbines out of the bay during low tide.

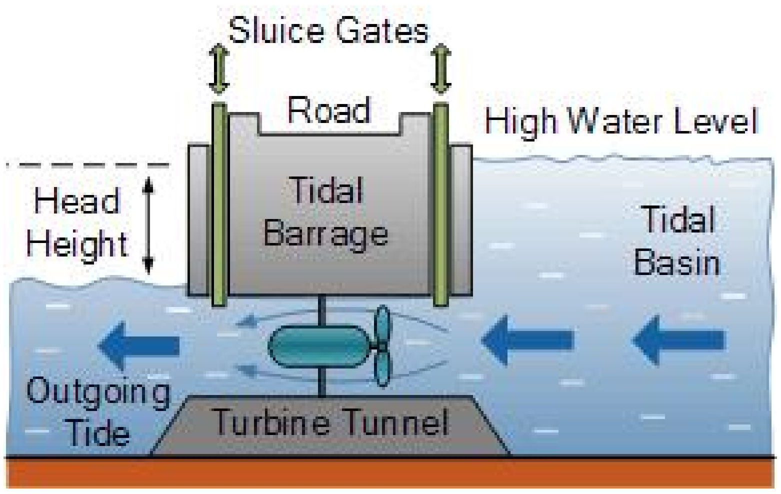


Figure 5.1: Cross-section of a tidal barrage during high tide [29]

Theoretically, the tidal barrage can use either one-way or two-way turbines (allowing power to be collected as the tide flows in *and* as the tide flows out). Neither method is objectively superior, however, as two-way turbines tend to cost more and can be less efficient than one-way turbines. The spectre of concerns other than expensive turbines is also present: from a cost standpoint, building the tidal barrage itself incurs substantial installation costs as an entire bay must be dammed off, after which years of constant maintenance fees are accrued. Additionally, there has been concern from an environmentalist perspective about the effects of damming a bay or inlet on the local ecosystem, though studies from the oldest tidal barrage in La Rance, France have demonstrated the environment's robust ability to recover, settling at an – albeit precarious and dependent on the continued operation of the tidal barrage – equilibrium [30] [31].

The second method of capturing tidal energy is quite similar to the idea of tidal barrages, though instead of relying on a natural bay or inlet, one can construct a man-made lagoon in the open water to simulate the same effects as a tidal barrage. Currently, a tidal lagoon is in development in Swansea, Wales.

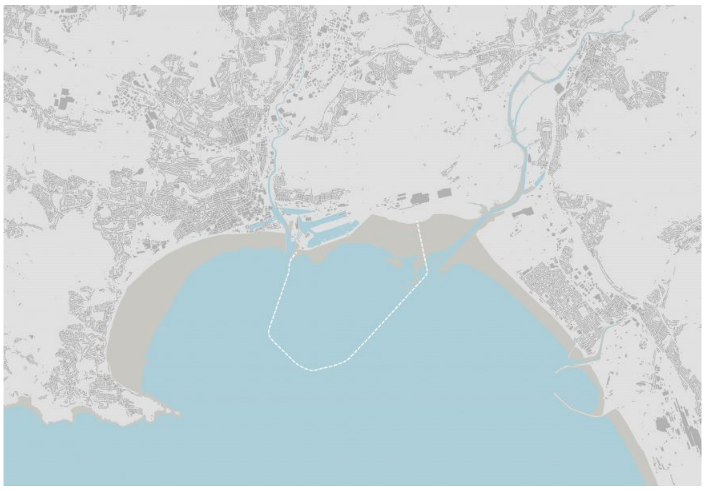


Figure 5.2: The outline of a man-made lagoon in Swansea, Wales (in construction) [34]

The third method of harvesting tidal energy was recently proposed by two Dutch coastal engineers. The as-of-yet untested idea, named Dynamic Tidal Power, is a natural extension of tidal lagoons to the open ocean and involves the creation of a massive causeway (with bi-directional tidal turbines embedded within) running parallel to the coast. These turbines are able to constantly generate power with the influx and efflux of the tides.

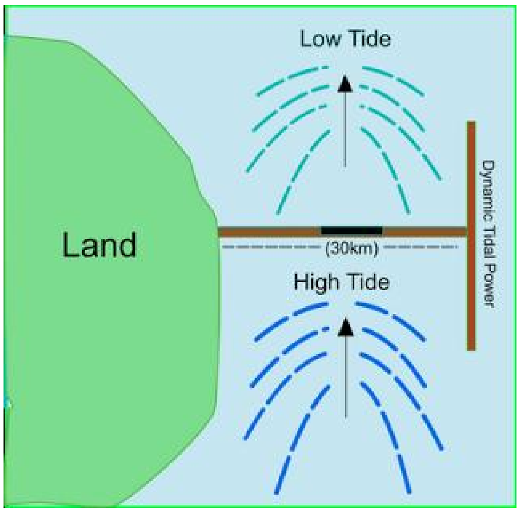


Figure 5.3: Dynamic Tidal Power (DTP) generation [33]

While Dynamic Tidal Power generation has not yet been proven, a massive project evaluating potential sites for DTP generation is underway in China [32].

The last method of generating tidal energy (and the one that will be considered in this chapter) is known as tidal stream generation. Under this paradigm, tidal turbines are anchored to the sea floor in certain areas where currents bottleneck due to natural (in theory, one could artificially create an impediment) obstruction – for instance, at the entrance of bays, rivers, between the shore and offshore islands, etc. — and tidal energy is constantly drawn from the subsurface currents brought about by natural tidal flows.

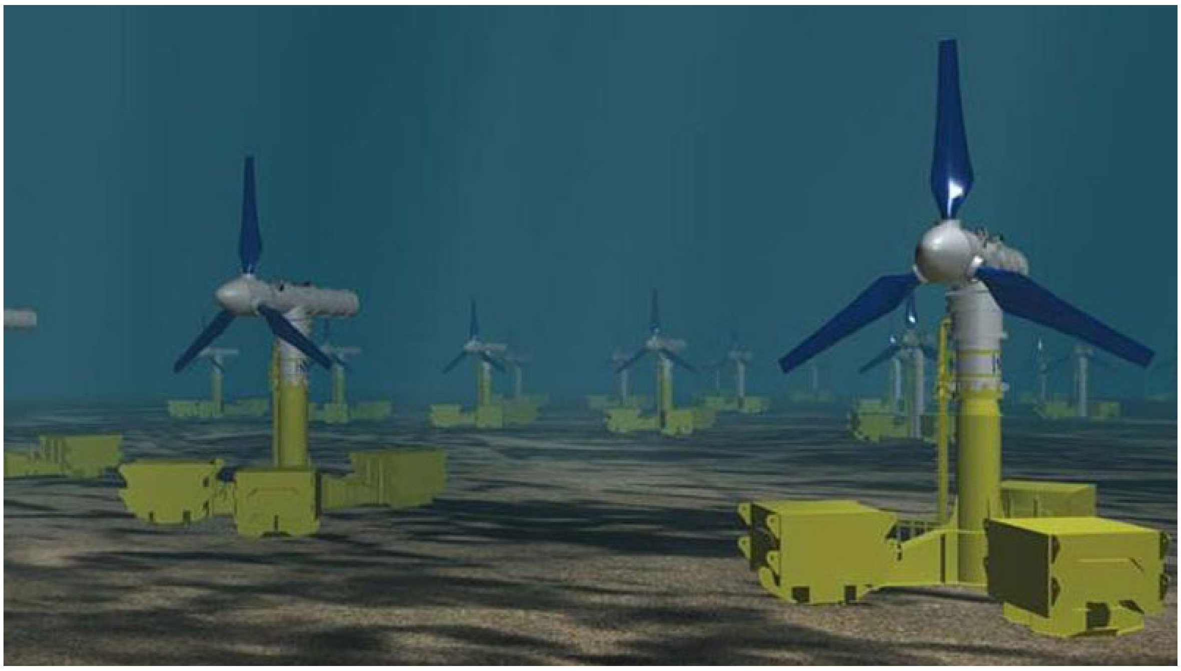


Figure 5.4: A field of tidal turbines (Tidal Stream Turbine generation) [35]

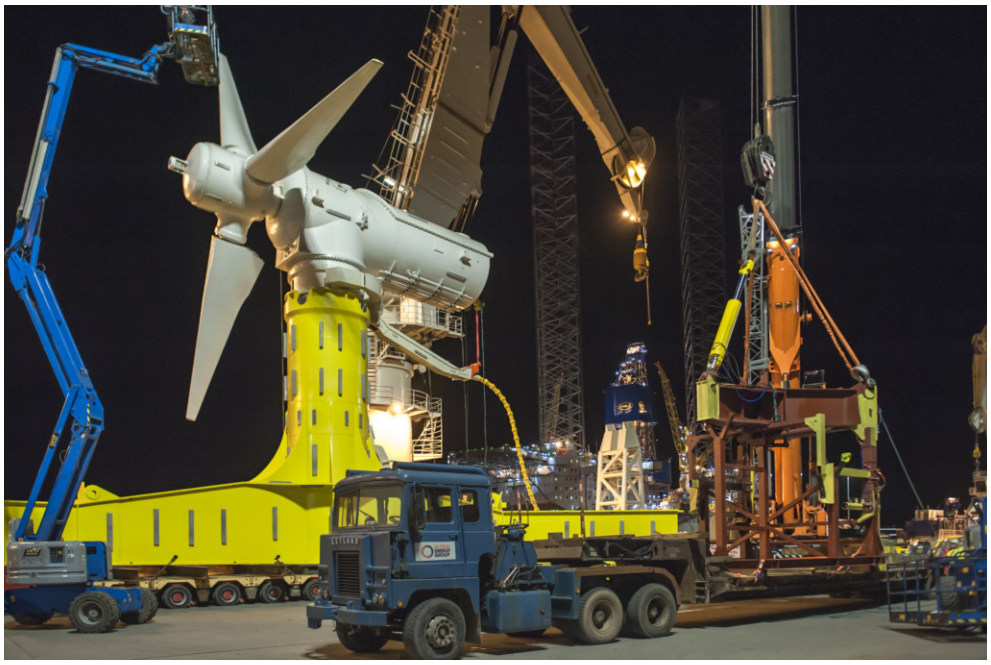
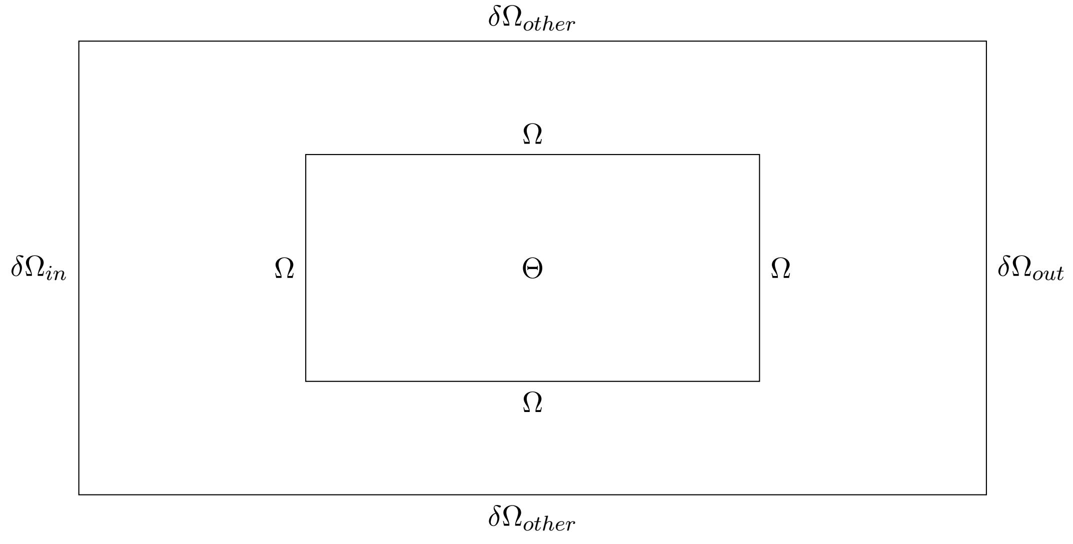


Figure 5.5: A tidal turbine destined for the Pentland Firth tidal array [36]

Tidal turbines have been increasing in size (and therefore power output) in the past decade as interest in tidal stream energy generation as increased. In Figure 5.4, we can see an idealized field of tidal stream turbines deployed on the sea bed. The following image of a turbine before its installation into the Pentland Firth tidal farm in Figure 5.5, however, gives a good sense of the scale of these turbines. Built by Atlantis Resources, this AR-1500 turbine has a 1500kW rating, and weighs 150 tonnes – while this may seem large, other turbines deployed in Orkney, Scotland, weigh over 500 tonnes. To summarize, tidal turbines are enormous, are built for a long lifetime (25+ years), and require a massive initial construction and installation cost, though these initial costs are projected to fall with increasing research and development of new tidal stream turbine technologies [37].

5.3 The Tidal Stream Turbine Problem Layout

In the tidal stream turbine optimization (TSTO) problem there is a two-dimensional or three-dimensional domain Ω in which the tidal turbines are placed. Generally there is also a sub-domain $\Theta \in \Omega$ such that the tidal turbines are all placed in Θ, with tidal modelling also occuring around the farm area in Ω. In the following examples, I consider unidirectional tidal currents in a 2-D rectangular domain. That is, the tidal velocity enters Ω from a boundary $\delta\Omega_{in}$ and flows out from $\delta\Omega_{out}$. There is assumed to be no tidal activity on the other two boundaries.



5.3.1 A Note on Tidal Cycles

As a PDE models the flow of water over time and two-dimensional space Ω, the choice of boundary condition is quite important. In the following scenarios, a constant velocity of water is assumed (i.e. water flows in $\delta\Omega_{in}$ at the same rate over the entire boundary at all times during the simulation). One can increase the complexity of the tidal inflow by using a sinusoidal model such as Funke et al. [38], or more complex models tailored to fit the actual natural region one is modeling (i.e. measure the tides at Pentland Firth and construct custom-made boundary conditions). Crucially, the assumption of a constant tidal flow allows us to remove time as a variable (because all points in time have the same exact inflow/outflow) in the PDE governing the movements of the tides.

5.4 Approaches in Tidal Stream Turbine Optimization (TSTO)

As aforementioned, tidal stream turbine optimization (TSTO) necessarily entails a trade-off: one can model the physics of the problem accurately using PDEs and produce a model which is computationally demanding to solve, or one can simplify the physics of the problem (using a non-PDE multi-wake model), and produce a solvable, but inaccurate result. Much of the optimization task, therefore, depends on the objective: how accurate of a solution do we need, and what are the runtime and memory constraints? Paired with these general goals, there exist a number of optimization methodologies (adjoint-based optimization, two-stage optimization, etc.) to mollify the complexity issue, which will be discussed in due time. These methods may been seen as an attempt to preserve the accuracy of the model by incorporating proper physics constraints, while intelligently choosing the formulation of the problem and the solution methodology in order to minimize computational complexity. For a comprehensive overview of TSTO modelling approaches, see the introduction of Funke et al.'s 2013 paper [40], or Funke et al.'s 2014 paper [39] which describes the continuum of options for modelling the tidal turbine problem in order to achieve specific goals; I will present here a cursory examination of previous methods.

Funke et al. have presented PDE-constrained optimization models for TSTO in several different papers, each of which incorporate an adjoint approach to design. Funke et al. show that if one uses an adjoint-based gradient approach (among other important specifications), a highly accurate and efficient model can be produced, one which incorporates the PDE constraints into the optimization problem, but is both computationally efficient *and* scalable to a high number of turbines. In their 2013 paper, they present a model in which they maximize the energy captured by the turbine farm (energy being a proxy for profit), subject to the shallow water equations (PDEs), the proximity constraints (the turbines must be a minimum distance away from each other), the boundary constraints (the turbines are placed inside a predefined domain), and the chosen number of turbines [40]. In a 2014 paper, the Barnett et al. use a simplified wake model instead of PDEs in order to globally optimize arrays of tidal turbines (noting this as fast, but inaccurate), and then implement a two-stage model in which the output of the cheap global optimizer is piped into the local gradient-based model which incorporates the PDE (the same model as in Funke et. al.'s 2013 paper), a process which they note improves both speed and accuracy in most cases [42]. In a 2016 paper, Funke et al. reconsider the TSTO problem by replacing the power maximization objective functional with an extensive cost-modelling functional and cable-routing optimization sub-problem (i.e. the problem morphs to minimizing the net present value of lifetime tidal turbine field cost rather than maximizing raw power) [41]. Finally, Funke et al. propose a continuous model which finds the most profitable field of tidal turbines while modelling the turbines as a continuous density function rather than discrete bumps. This last paper both incorporates cost into the objective functional and does not require the authors to pre-specify the number of turbines to be placed. Additionally, the authors show that the results from the continuous model largely agree with those of the discrete model from the 2013 paper [38]. It is noted, however, that the methodology of converting a continuous density field into optimal discrete turbine locations is a simple stochastic method which I will endeavour to improve upon later in this chapter.

Other formulations of TSTO (or similar wind farm optimization problems) use simpler equations to model the physics of the problem. These formulations, as Funke et al. note:

> [They] simplify the tidal flow model such that the solutions are either available as explicit analytical expressions, or are extremely fast to compute... While this approach can provide a coarse estimate for the power potential of a site, these simplified models cannot accurately capture the complex nonlinear flow interactions between turbines. [40]

On the most-simplifying end of TSTO and TSTO-like models are mixed-integer programs. Though to my knowledge no purely MIP model exists for tidal turbine optimization, there are multiple papers which use MIP models for wind turbine optimization. Zhang et al. propose a MIP model for optimizing the placement of a given number of turbines in a fixed n x n grid domain, subject to the relevant aerodynamic (wind and wake) constraints. They create models to maximize the expected power of the wind farm and the average power captured at each grid point labelled by $i \in 1, ..., n^2$. Yet these models incorporate the physics of the problem as analytical multi-wake equations *not governed by PDEs*, which are baked into the objective function or added as constraints. The decision variables are integer variables corresponding to the choice of placement of the fixed number of turbines. Simplifying greatly, with predetermined number of turbines k, energy = E, energy per grid unit = E_i, the variable x_i being a binary variable taking value 1 if a turbine is placed at i and 0 otherwise, and $\mathcal{N}(i)$ a set of all points proximal to any given location i (the set is determined by the chosen grid and proximity constraints), their model takes the form:

$$\max\ E \quad s.t. \quad \begin{cases} \sum_{i=1}^{n} x_i = k \\ x_i + x_j \leq 1 \quad \forall j \in \mathcal{N}(i) \\ x_i \in \{0,1\} \quad \forall i \in 1,...,n \end{cases} \tag{5.1}$$

It is noted by Zhang et al. that their initial optimization problem is nonlinear in the objective, and thus they construct more complex and accurate models to which they add a host of constraints, allowing for linearization of the objective function. I will not discuss this further, however, as I will be focusing on tidal turbines. For more information on the specific models, see Zhang et al. [43]. For our purposes, the most important takeaway from their paper lies in modeling each possible turbine location with a binary variable; their MIP algorithm then finds which binary variables to turn on and off, subject to the required number (or maximum number) of activated turbines. This method is similar to the Source Inversion problem (in which the optimization algorithm must pick which source functions to activate and which to set to zero), and is a decision methodology I will try to incorporate into a MIPDECO formulation of the TSTO problem in a later section.

The final paradigm used in wind turbine optimization is exemplified by Fagerfjall, who creates a similar model to Zhang et al. for optimizing wind turbine layout using a MIP model, albeit with a far simpler objective function and more constraints [44]; in this case, the optimization algorithm is neither MINLP nor gradient-based, but is a heuristic global search, which would be infeasible in any optimization problem containing PDEs for the aforementioned reasons.

5.5 The Shallow Water Equations

In the following sections, I will examine a continuous and discrete approach to TSTO, and attempt my own formulation of a continuous-to-MIPDECO optimization pipeline. In each of these scenarios, the physics of the TSTO problem will be approximated using the shallow water equations, following a series of papers by Funke et. al. [38] [40] [41] [42]. A cursory explanation of the shallow water equations as applies to the TSTO problem follows.

The motion of viscuous fluid substances may be accurately described by the nonlinear PDE known as the Navier-Stokes equations. Unfortunately, the Navier-Stokes equations are analytically intractable (existence and smoothness of N-S solutions are considered one of the seven most important unsolved problems by the Clay Mathematics Institute). Consequently, the Navier-Stokes equations are approximated by the two-dimensional shallow water equations, which are derived by depth-integrating the Navier-Stokes equations. As Randall notes:

> The shallow water equations are the simplest form of the equations of motion that can be used to describe the horizontal structure of an atmosphere. They describe the evolution of an incompressible fluid in response to gravitational and rotational accelerations. The solutions of the shallow water equations represent many types of motion, including Rossby waves and inertia-gravity waves [45].

While the shallow water equations are a set of nonlinear PDEs, it is noted that a more realistic model could incorporate even more complex PDEs such as the Reynolds-averaged Navier-Stokes equations (RANS) and three-dimensional Actuator Disk Momentum (ADM) theory. Though it seems work on integrating Funke's work and open-source code OpenTidalFarm with ADM theory and the Fluidity framework is ongoing, the scale of the integration is regrettably out of the scope of this project [40]. The interested reader can refer to an excellent overview of RANS and ADM applied to the TSTO problem given by Abolghasemi et al. [46].

In the following sections, the shallow water equations are used with parameters described in the table below. The TSTO problem occurs in domain Ω. Each turbine is modelled as a smooth differentiable bump which represents the increased bottom friction resulting from the existence of a turbine centered at a given pair of two-dimensional coordinates. The decision variable $\boldsymbol{m}$ is the vector of these turbine coordinates, $x_1, y_1....x_N, y_N$, which can include friction coefficients $K_1...K_N$ if the coefficients are individually tuned to the turbines (or alternatively a single value of K can be ubiquitous).

The shallow water equations have several parameters. First, $\kappa \in \{0, 1\}$ is a binary variable which determines stationarity. If $\kappa = 0$, then the velocity of flow (that is, the tidal currents) are considered stationary over time. If $\kappa = 1$, we allow the tidal currents to change over the time simulation period. Assuming time-invariance for now, $\boldsymbol{u}$ represents the depth-averaged velocity of water flow at any given point in the domain, and thus $\boldsymbol{u} : \Omega \to \mathbb{R}^2$; note that $\boldsymbol{u}$ is a two-dimensional vector, which will have important ramifications for the finite element method later on. The next parameter is the free-surface displacement, $\eta : \Omega \to \mathbb{R}$. Acceleration due to gravity is accounted for by $g \in \mathbb{R}$. The resting water depth is given as $H : \Omega \to \mathbb{R}$ (which also satisfies $h = \eta + H$, where h is the total water depth), and the constant natural background friction $c_b : \Omega \to \mathbb{R}$. The farm-induced friction flow (i.e. the artificial friction which the turbines induce on the natural current) is given by $c_t : \Omega \to \mathbb{R}$, and finally, the kinematic viscosity term $\nu \in \mathbb{R}$, while eddy viscosity is a fixed constant which does not appear.

There is a Dirichlet boundary condition applied to $\boldsymbol{u}$ on the inflow boundary $\delta\Omega_{in}$, with $g(.)$ a predefined function of $\boldsymbol{u}$. For instance, $g(.)$ can be chosen to represent sinusoidal tidal movement, or $g(.)$ could return a vector of constants specifying a constant time-invariance water flow velocity. Free-surface displacement is set to zero on the outflow boundary $\delta\Omega_{out}$. A strong no-slip boundary condition is imposed on $\boldsymbol{u}$ on the two other boundaries $\delta\Omega_{other}$. The shallow water equations are presented below in 5.2 and 5.3, along with the boundary conditions in 5.4.

Parameter	Description
t	$t \in (0, T)$, the simulation period
$\boldsymbol{m}$	Locational decision variables
Ω	Turbine farm state space
ρ	Fluid (water) density
$\boldsymbol{u}$	Velocity of the flow
η	Free surface displacement
g	Acceleration due to gravity
ν	Kinematic viscosity
c_b	Constant background bottom friction
c_t	Enhanced friction of parametrized turbines
H	Resting water depth
κ	0 if stationary, else 1

$$\kappa\frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + g\nabla\eta + \frac{c_b + c_t(\boldsymbol{m})}{H}||\boldsymbol{u}||\boldsymbol{u} = 0 \tag{5.2}$$

$$\kappa\frac{\delta \eta}{\delta t} + \nabla \cdot (H\boldsymbol{u}) = 0 \tag{5.3}$$

$$\text{Boundary Conditions} = \begin{cases} \boldsymbol{u} = g(\boldsymbol{u}) & \boldsymbol{u} \in \delta\Omega_{in} \\ \eta = 0 & \eta \in \delta\Omega_{out} \\ \boldsymbol{u} = \boldsymbol{0} & \boldsymbol{u} \in \delta\Omega_{other} \end{cases} \tag{5.4}$$

5.6 TSTO Formulated as a Discrete Problem

Funke et al. approach TSTO as a PDE-constrained optimization problem which approximates the physics of the situation by the two-dimensional shallow water equations (assuming boundary equations in the previous section) and tidal turbines by bumps of increased friction. It is important to note that the number of turbines N to place in the domain Ω must be predefined, and a minimum distance between any two turbines d is introduced (typically a function of turbine blade diameter or turbine length).

$$\kappa\frac{\delta u}{\delta t} + u \cdot \nabla u - \nu\nabla^2\boldsymbol{u} + g\nabla\eta + \frac{c_b + c_t(\boldsymbol{m})}{H}||u||u = 0 \tag{5.5}$$

$$\kappa\frac{\delta\eta}{\delta t} + \nabla \cdot (H\boldsymbol{u}) = 0 \tag{5.6}$$

In the following I assume stationarity (that is, the average power extracted from each turbine does not vary over time, and thus $\kappa = 0$ and time can be eliminated as a parameter of interest in this situation). Each turbine is parametrized as a two-dimensional bump resulting from the multiplication of the below bump equation in each dimension:

$$\psi_{p,r} = \begin{cases} e^{1-1/(1-||\frac{x-p}{r}||)} & ||\frac{x-p}{r}|| < 1 \\ 0 & otherwise \end{cases} \tag{5.7}$$

Given a friction coefficient K_i, the friction function of each turbine with a center at x_i, y_i is given by:

$$C_i(m)(x, y) = K_i \cdot \psi_{x_i,r}(x) \cdot \psi_{y_i,r}(y) \tag{5.8}$$

Then in the above PDE, $c_t(\boldsymbol{m}) = \sum_{i=1}^{N} C_i(m)$. The objective functional is given by the average power extracted from the field:

$$P(\boldsymbol{m}) = \int_{\Omega} \rho c_t(\boldsymbol{m})||\boldsymbol{u}||^3 \mathrm{d}x \tag{5.9}$$

The PDE-constrained optimization problem is written below. Note that the last constraint ensures that a minimum distance d exists between all turbine locations.

$$P(\boldsymbol{m^*}) \equiv \max_{m} \; P(\boldsymbol{m}) \quad s.t. \quad \begin{cases} \boldsymbol{u} \cdot \nabla\boldsymbol{u} - \nu\nabla^2\boldsymbol{u} + g\nabla\eta + \frac{c_b + c_t(\boldsymbol{m})}{H}||\boldsymbol{u}||\boldsymbol{u} = 0 \\ \nabla \cdot (H\boldsymbol{u}) = 0 \\ ||p_i - p_j||_2^2 \geq d^2 \quad 1 \leq i < j \leq N \end{cases} \tag{5.10}$$

This model can then be solved using a gradient-based approach similar to that of the Source Inversion problem, again incorporating the adjoint approach to design, as described in the Appendix. As with the Source Inversion problem, the PDE constraint must first be discretized by finding the solution to the weak formulation of the system of equations. Yet a key difference exists between the two problems, as in Source Inversion problem, $\boldsymbol{u} : \Omega \to \mathbb{R}$. Consequently, it was sensible to use piecewise *linear* finite elements to represent $\boldsymbol{u}$ over its discretized domain. In the TSTO problem, free surface displacement η will similarly be represented by elements drawn from the space of all piecewise linear functions. As the depth-averaged velocity $\boldsymbol{u} : \Omega \to \mathbb{R}^2$, however, $\boldsymbol{u}$ will now be represented by finite elements drawn from the space of all piecewise *quadratic* functions. For the mathematics of the finite element method as applied to this problem, see the Appendix.

5.6.1 A Solved Discrete TSTO Example

Suppose we want to optimize the placement of 32 tidal turbines in a rectangular domain Ω following Funke et al. [40] closely, with some parameter modifications as stated and justified. For the purposes of this toy example, we assume the farm is placed in the rectangular sub-domain Θ nested in the center of the domain (this makes handling boundary conditions easier). Ω is assumed to be a $800m$ x $600m$ domain with mesh vertices $30m$ apart outside the farm area, and $3m$ apart inside the farm area (with mesh construction occuring in open source software gmsh). The ν, H, c_b and K (constant across turbines) were adapted from Funke et. al [40], while gravity and water density assume their usual values.

The Atlantis AR-1500 tidal turbine (15 meters tall and 16 meters in blade diameter, minimum deployment depth of $30m$ [37]) which was recently placed in Pentland Firth is used as an example for turbine. Consequently, I use a blade diameter of $16m$ (radius $r_b = 8$) and minimum distance $d = 20m$ between turbines. The boundary conditions hold as aformentioned, with a constant horizontal velocity of $2m/s$ on the inflow boundary so that $g(\boldsymbol{u}) = \begin{bmatrix} 2 & 0 \end{bmatrix}$ $\boldsymbol{u} \in \delta\Omega_{in}$.

Parameter	Description
Domain Ω	$600m$ x $800m$
Farm Area Θ	$400m$ x $200m$
Water Density ρ	$1000kg/m^3$
Kinematic Viscosity ν	$3m/s$
Gravity g	$9.81m/s^2$
Constant Background Friction c_b	0.0025
Turbine Friction Coefficient$K$	21
Turbine Blade Radius r_b	$8m$
Resting Water Depth H	50m
Velocity of Flow $\boldsymbol{u}_{in}$	$2m/s$

Figure 5.6: Discrete TSTO Parameters

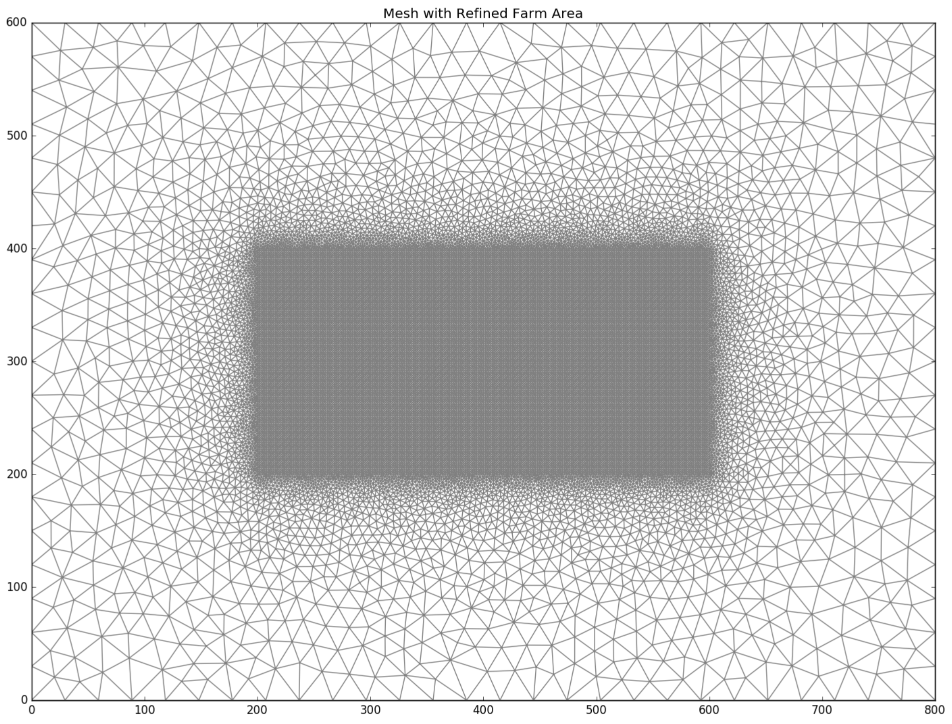


Figure 5.7: The 800 x 600 mesh Ω with a refined farm area Θ

Given the above mesh and parameters, 32 turbines are created as friction bumps. In this initial configuration, 30.2 megawatts (MW) of power are extracted, or .94375 MW per turbine.

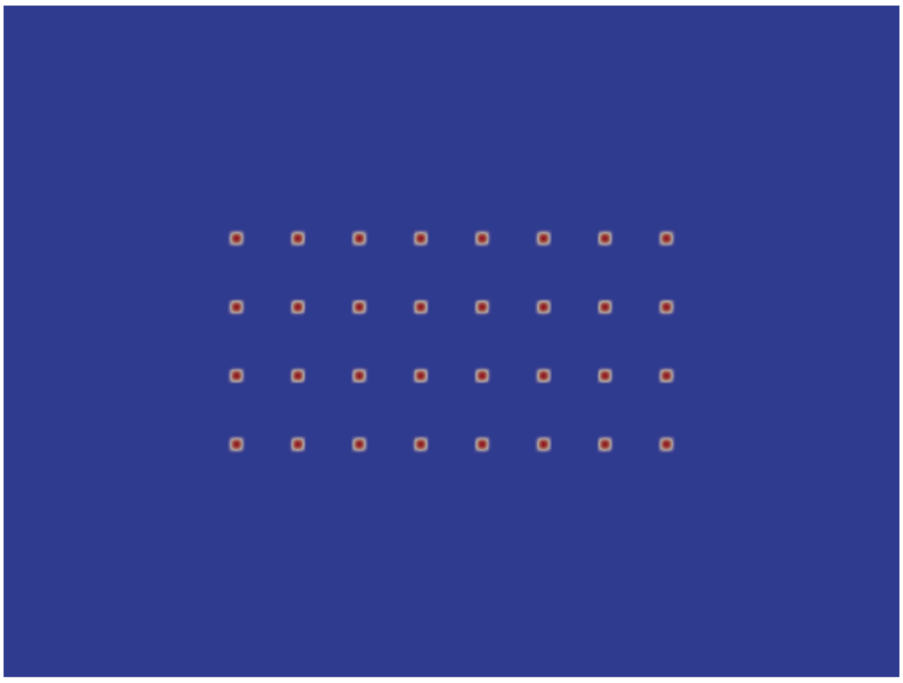


Figure 5.8: Initial layout of turbines

With the minimum distance between turbines as $d = 20$, I used SLSQP in SciPy to optimize the placement of the 32 turbines in the area of the farm, which converged after 82 iterations with a tolerance of $1e-6$. In this final layout, 47.6 MW of power are extracted, or 1.49 MW per turbine, a 54% increase in power!

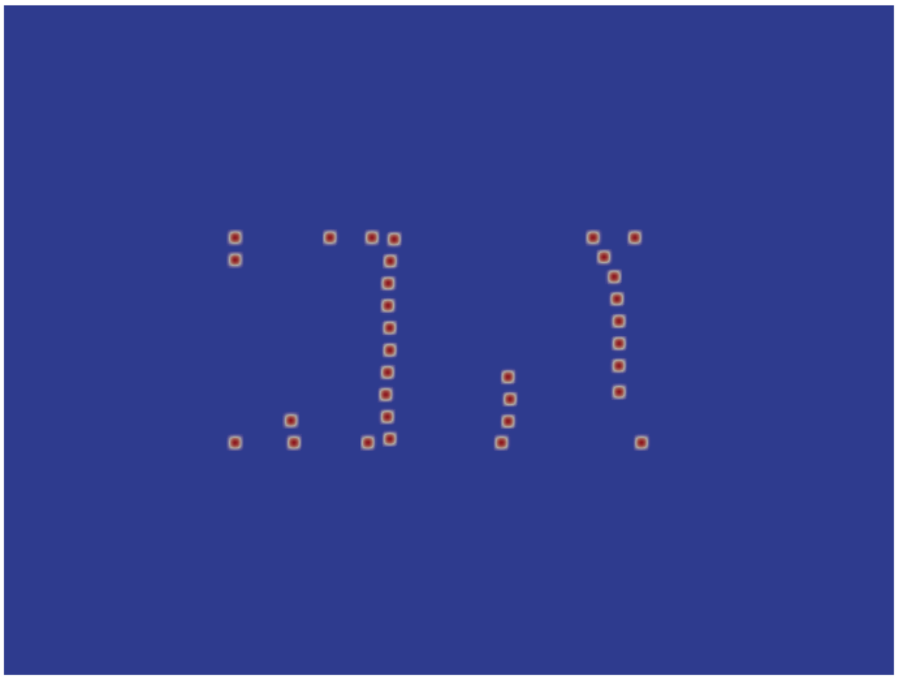


Figure 5.9: Optimized layout of turbines

5.7 TSTO Formulated as a MIPDECO Problem

There are a few principal shortcomings of the above model. The first is that is uses the shallow water equations and simple friction bumps for turbines instead of the more accurate ADM theory, a topic that will not be treated here. Another issue is that the precise number of turbines to be placed must be pre-specified. That is, exactly N turbines are coded in $2 \cdot N$ coordinates – there is no option to place more or less than N turbines. This is a solution which is easily remediable using a MIPDECO approach, or the continuous approach, which is explored in a later section. Next, the objective functional attempts to maximize power, an imperfect proxy for profit. Later papers by Funke et al. incorporate an extremely complicated explicit profit function derived from an optimal cable-routing sub-problem, as well as a simple profit function which is reliant on a predefined profit margin; I will soon present a model which attempts to take a middle ground in presenting a fairly simple profit function which still takes into account time discounting and power prices with no *a priori* profit margin. The last issue is the constant bane of gradient-based optimization: any solution will necessarily be a local minima. Funke et al.'s later two-step global to local optimization algorithm, also discussed in a later section, is a solution to this.

It is important to note that allowing variation in the chosen number of turbines and analyzing profit instead of power inherently changes the optimization problem and objective functional. If we maintain the objective functional as power extracted, then it is likely that the optimal number of turbines is simply the total number of turbines which can safely fit in the farm domain, as the marginal power increase from adding one turbine, however minute, will almost always be positive unless including so many turbines that they crowd each other's power generation out. In reality, this marginal increase in power will be offset by a somewhat-fixed marginal cost of placing a new turbine and the variable marginal cost of adding a new turbine to the network.

To avoid a spurious optimization problem, we can change the objective functional to average *profit* extracted from the array, rather than average power. Note that time must be included even if stationary power extraction still holds, as net present value and its relation to profit is inherently non-stationary. For now, I will present a high-level MIPDECO approach assuming a predefined profit function, which will be presented in a later section.

Consider Funke et al.'s optimization formulation, which we can tweak into a MIPDECO. First, we can discretize the continuous domain Ω (which is necessary anyhow to solve using any gradient-based method) into a n x n mesh (I will continue to call this Ω), and assign a binary variable to each mesh vertex. All of the parameters from section 5.4 remain the same. Given a friction coefficient K_i, we now depart from Funke at add binary variables to each turbine's friction function. Thus the friction function of each turbine with a center at x_i, y_i is given by:

$$C_i(w_i) = w_i \cdot K_i \cdot \psi_{x_i,r}(x) \cdot \psi_{y_i,r}(y) \tag{5.11}$$

In the shallow water equations, we now have $c_t(\boldsymbol{w}) = \sum_{i=1}^{n^2} C_i(w_i)$. The objective functional is given by the average power extracted from the field – note that this is now a function of the binary controls, rather than turbine location (note that technically the integral changes to a summation operator due to discretization):

$$P(\boldsymbol{w}) = \int_0^T \int_\Omega Profit(\boldsymbol{w}) \mathrm{d}x \mathrm{d}t \tag{5.12}$$

Similarly to the Source Inversion problem, we need to pick up to N sources to turn on in order to maximize the energy captured. Note the implicit assumption that each mesh vertex is d units apart, undertaken so that the last constraint ensures that any activated turbine has no other active turbines among its neighboring mesh vertices (denoted $\mathcal{N}(.)$), which are distance d away.

$$\max_{\boldsymbol{w}} \; P(\boldsymbol{w}) \quad s.t. \quad \begin{cases} \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + g \nabla \eta + \frac{c_b + c_t(\boldsymbol{w})}{H} ||\boldsymbol{u}|| \boldsymbol{u} = 0 \\ \nabla \cdot (H \boldsymbol{u}) = 0 \\ \sum w_{xy} \leq N \\ w_{xy} \in \{0, 1\} \;\; \forall x, y \in \Omega \\ w_{xy} \cdot w_{ab} = 0 \;\; \forall x, y \in \Omega, \forall a, b \in \mathcal{N}(x, y) \end{cases} \tag{5.13}$$

At first glance, this formulation may seem promising, as it directly optimizes for profit without requiring a prespecified number of turbines (just an upper bound). Unfortunately, this MIPDECO formulation is restricted in that a mesh fineness must be set so that each binary variable is exactly d meters apart. Naturally, if one increases the size of the domain (i.e. increasing the domain from $800m$ x $600m$ to $2000m$ x $2000m$), then so too does the number of binary variables increase, as this formulation has exponentially mesh-dependent variables. Consequently, any computation with a realistic domain will be nearly impossible (as opposed to the Source Inversion problem, the meshes in this scenario need to be far larger than 64 x 64 as our actual domain will be hundreds of meters by hundreds of meters). As a result, a direct MIPDECO formulation is out of the cards. What is possible, however, is to solve this problem in a continuous fashion or two-step fashion which will be enumerated upon in the following sections. After a solution is obtained, the near-optimal solution can be converted into a mesh-independent MIPDECO and solved for enhanced accuracy. This can be done, for instance, by creating an optimal turbine density field and then converting the largest $2N$ or so hotspots into turbines with corresponding binary variables, which can then be fed into a MIPDECO problem.

5.8 TSTO Formulated as a Continuous Problem

Along with the discrete TSTO framework, Funke et al. present a continuous model which optimizes a turbine density field, as opposed to individually resolving the locations of a predetermined number of turbines [38]. The continuous TSTO problem attempts to maximize the *profit* of a farm with domain Ω over its lifetime as a function of parameter $d : \Omega \to \mathbb{R}$, which measures turbine density at any location $\boldsymbol{x}$ on the farm. The value of $d(\boldsymbol{x})$ is upper bounded by a maximum turbine density $\bar{d}$, which is 0 where no turbines can be installed (areas that are too steep, too shallow, have natural physical impediments, etc.). The maximum turbine density is defined as $\bar{d} = 1/min\_dist^2$, where min_dist is the minimum distance required between turbines if they were discrete and individually resolved. By using $d(\boldsymbol{x})$ as a control, this method is able to automatically find the optimal number of turbines by integrating the density over the domain Ω. That is, for optimized turbine density $d^*(\boldsymbol{x})$, one can find the optimal number of turbines N:

$$N = \int_\Omega d^*(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \tag{5.14}$$

The objective functional is modelled as:

$$Profit(d) \equiv Revenue(d) - Cost(d) = IkE(u, d) - C\int_\Omega d(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \tag{5.15}$$

In this formulation, I is the income per energy unit, k a coefficient between 0 and 1 which accounts for efficiency losses over the lifetime of the farm, and E is the total amount of energy produced by the farm over its lifetime given lifetime velocity u, and d is the turbine density. C accounts for fixed and variable costs of a single turbine over its lifetime. The shallow water equations again model the physics of the situation, though the continuous formulation and profit necessitates the use of the time-variant formulation and turbine friction is a function of the density field rather than discrete turbine locations.

$$\frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} - \nu\nabla^2\boldsymbol{u} + g\nabla\eta + \frac{c_b + c_t(d)}{H}||\boldsymbol{u}||\boldsymbol{u} = 0 \tag{5.16}$$

$$\frac{\delta\eta}{\delta t} + \nabla\cdot(H\boldsymbol{u}) = 0 \tag{5.17}$$

Correspondingly, while most parameters stay the same as in the discrete formulation, in the time-invariant case, it is assumed that the depth-averaged velocity $\boldsymbol{u} : \Omega \text{ x } (0, T) \to \mathbb{R}^2$, and the free surface displacement $\eta : \Omega \text{ x } (0, T) \to \mathbb{R}$.

In order to use d as the control, it is necessary to derive a relationship between the artificial friction generated by the farm of turbines at time t, c_t, and turbine density d. The authors note that:

> To derive the relationship between the farm induced friction c_t and the turbine density function d, we use an idea very similar to the enhanced bottom drag formulations (Divett et al., 2013; Funke et al., 2014; Martin-Short et al., 2015), where the turbine induced drag is chosen such that the resulting force approximates the drag force of an analytical model of the turbines [38].

Using the depth-integrated momentum equation obtained from the shallow-water equations, Funke et. al model the force produced by the farm as

$$\boldsymbol{F}_{farm}(t) = \int_\Omega \rho c_t(d(\boldsymbol{x}))d(\boldsymbol{x})||\boldsymbol{u}(\boldsymbol{x},t)||\boldsymbol{u}(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} \tag{5.18}$$

They then consider each turbine with 3-dimensional flow, upstream velocity $||\boldsymbol{u}||_\infty$, and with C_T being the constant drag coefficient of each turbine and A_T being the constant surface area of the blade-span of each turbine (both of which are assumed constant across all turbines), and model the force of each turbine (and the force of the farm as just the sum of N individual turbine forces) as:

$$\boldsymbol{F}_{turbine}(\boldsymbol{u}_\infty) = \frac{1}{2}\rho C_T A_T ||\boldsymbol{u}||_\infty \boldsymbol{u}_\infty \tag{5.19}$$

$$\boldsymbol{F}_{farm} = \sum_{i=1}^{N} \frac{1}{2}\rho C_T A_T ||\boldsymbol{u}||_\infty \boldsymbol{u}_\infty \tag{5.20}$$

Correspondingly, they parametrize the farm drag force as the integral of continuous drag scaled by the turbine density function d, so that:

$$\boldsymbol{F}_{farm}(t) = \int_\Omega d(\boldsymbol{x}) \cdot \frac{1}{2}\rho C_T A_t ||\boldsymbol{u}(\boldsymbol{x},t)||\boldsymbol{u}(\boldsymbol{x},t) d\boldsymbol{x} \tag{5.21}$$

They obtain $c_t(d(\boldsymbol{x}))$, then, from comparing Equations 5.18 and 5.21, and making the appropriate substitutions.

$$c_t(d(\boldsymbol{x})) = \frac{1}{2} C_T A_T \mathrm{d}(\boldsymbol{x}) \tag{5.22}$$

The power extraction of the farm at time t is given by the dot product of the force of the farm and the depth-averaged velocity, and is integrated over the lifetime of the farm, yielding:

$$E(\boldsymbol{u}, d) = \rho \int_0^T \int_\Omega c_t(d(\boldsymbol{x})) ||\boldsymbol{u}(\boldsymbol{x},t)||_2^3 \mathrm{d}\boldsymbol{x}\mathrm{d}t \tag{5.23}$$

Finally, the authors divide the objective functional by TIK, noting scale-invariance of this optimization paradigm. This yields the continuous PDE-constrained optimization problem:

$$\max_d \frac{1}{T} E(\boldsymbol{u}, d) - \frac{C}{TIk} \int_\Omega d(\boldsymbol{x}) \mathrm{d}x \quad s.t. \quad \begin{cases} \frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + g \nabla \eta + \frac{c_b + c_t(d)}{H} ||\boldsymbol{u}||\boldsymbol{u} = 0 \\ \frac{\delta \eta}{dt} + \nabla \cdot (H\boldsymbol{u}) = 0 \\ 0 \leq d(\boldsymbol{x}) \leq \bar{d}(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \Omega_{farm} \\ d(\boldsymbol{x}) = 0 \quad \forall \boldsymbol{x} \in \Omega \setminus \Omega_{farm} \end{cases} \tag{5.24}$$

To estimate the cost coefficient, the authors use a tidal cycle with peak velocity $\boldsymbol{u}_{peak}$ (note that with constant tidal flow, $\boldsymbol{u}_{peak}$ is the constant tidal flow; if using sinusoidal tidal flow, however, $\boldsymbol{u}_{peak}$ is the peak velocity of the tidal period) and profit margin $m = \frac{Revenue - Cost}{Revenue}$, the authors estimate:

$$\frac{C}{TIk} = \frac{1}{2} C_T A_T (1-m) \rho \boldsymbol{u}_{peak}^3 \tag{5.25}$$

An important point to note is that $\frac{C}{TIk}$ is extremely easy to calculate and becomes a constant – *when the user defines the profit margin.* Thus one is able to ignore many difficult parameters required to make an accurate profit calculation, at the expense of making the profit function entirely dependent on a user-specified profit margin. While this certain cleans calculations up, the profit margin varies greatly between situations and estimating a margin is quite difficult due to the nascency of tidal stream power generation and the related lack of historical information regarding the operations of tidal turbine farms.

This model can be solved in a fashion similar to the discrete case, using OpenTidalFarm and gradient-based optimization. As there are no inequality constraints, L-BFGS-B is an ideal optimization algorithm, as its limited-memory properties reduce the stress of storing large Hessian approximations in the computer's memory. As aformentioned, the output of this model is $d^*(\boldsymbol{x})$, the optimal turbine density function. Funke et al. use a probabilistic algorithm to convert this continuous density field into its discrete approximation, first by integrating the optimal field over the domain to find the number of discrete turbines, before micro-siting the turbines [38]:

Algorithm 12 Transforming a continuous density field to discrete turbine farm

```
1: Determine the number of turbines, $N = \int_\Omega d^*(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$
2: while number of turbines deployed is less than $N$ do
3:     Pick a random point $\boldsymbol{x}$ in the domain
4:     Place a turbine at $\boldsymbol{x}$ with probability $d(\boldsymbol{x})/\bar{d}$ if the distance constraint is satisfied
5: Return optimal discrete field of turbines
```

Though I will not further examine the derivation of continuous model, one can refer to sections 2, 3, and 4 of Funke et al.'s recent paper for a detailed mathematical derivation and justification [38]. While I will implement this model as a baseline solution, my intention is to use it as the starting point of my model in the next section, which will incorporate time-discounting and actual cost and revenue estimates (as opposed to a predefined profit margin). More importantly, I intend to replace the heuristic process which converts the density field to individually resolved turbines by introducing a targeted MIPDECO problem that will take the optimal density field as its initial input.

5.8.1 A Solved Continuous TSTO Example

If we take the continuous optimization problem in Equation 5.24, we can see that it is essentially an extension of the discrete case. The shallow water equations have been augmented to include variation in the tides over time, and the decision variables are now a continuous density field rather than discrete bumps. The former objective functional (power extraction) is now a component on the new objective functional, which models profit as a function of cost and revenue (which depends on power). Let us consider Equation 5.24 using the same domain Ω and parameters for H, ν, g, ρ, r_b, and c_b as in the discrete case. As in the discrete case, we assume a simple tidal flow of $2m/s$. Consequently, we may ignore the time parameters as the velocity of the flow is time-invariant, and no discounting is applied to the profit function. Thus we can simply solve the same steady-state shallow water equations as in the solved discrete example. In order to estimate a profit over the lifetime of a farm, this model would simply scale profit by the appropriate time period (for instance, if we assume $\delta t = 1$, total profit from $t = 0$ to $t = 5$ would simply be $5 \cdot Profit|_{t=0}$). For modelling purposes, time invariance can be realized by ignoring the time derivative terms in the shallow water equations, and setting $T = 1$. Before running the optimization problem, we can calculate the cost coefficient C by defining the turbine blade radius $r_b = 8m$ and finding the area swept out by the turbine blades $A_T = 2\pi r_b = 201.06m^2$. Funke et al.'s baseline parameters for the profit margin $m = 40\%$ and and thrust coefficient $C_t = .6$ are used, yielding a cost coefficient value of $289.5kW$

Parameter	Description
Domain Ω	$600m$ x $800m$
Farm Area Θ	$400m$ x $200m$
Water Density ρ	$1000kg/m^3$
Kinematic Viscosity ν	$3m/s$
Gravity g	$9.81m/s^2$
Constant Background Friction c_b	0.0025
Turbine Blade Radius r_b	$8m$
Thrust Coefficient C_T	$0.6m$
Cross-Sectional Turbine Surface Area A_T	$201.06m^2$
Cost Coefficient $\frac{C}{TIk}$	289.5 kW
Resting Water Depth H	50m
Velocity of Flow $\boldsymbol{u}_{in}$	$2m/s$

Figure 5.10: Continuous TSTO problem parameters

After setting up the optimization problem with the above parameters and domain, I used L-BFGS-B with a tolerance of $1e-6$. Converging after 47 iterations, the following optimal density field was produced, with $N = 53$ (rounded down from 53.2) being the optimal number of turbines obtained by integrating $d^*(\boldsymbol{x})$ over Ω. 20.06 MW of power was produced at a cost of 15.4 MW, yielding 4.7 MW of power in profit. Note that the red regions in Figure 5.11 correspond to the highest-density of turbines, while blue regions correspond to the lowest turbine density.

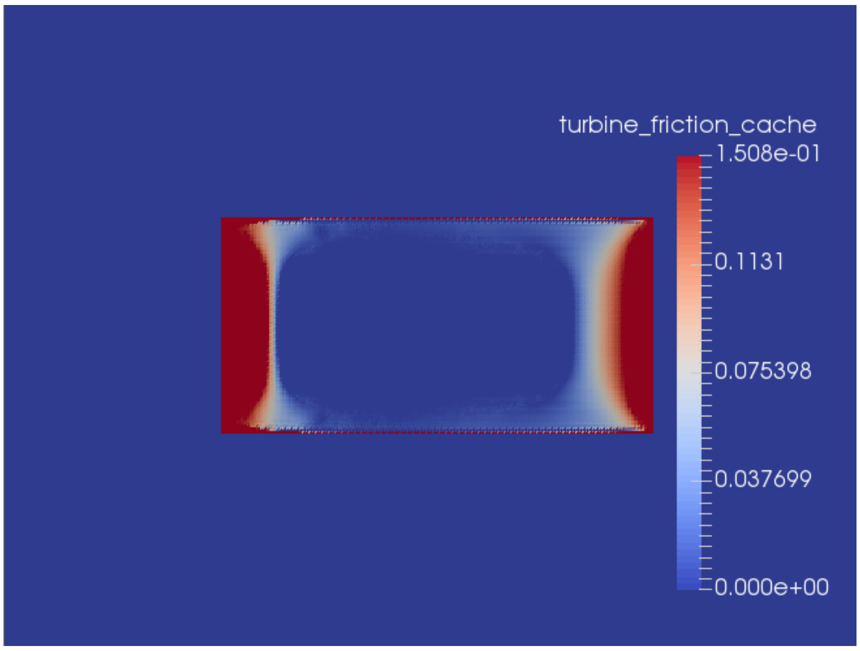


Figure 5.11: Optimal turbine density field

Using Funke et al.'s conversion method (algorithm 12), I extracted coordinates of the first N turbines placed and create a discrete turbine field.

To estimate the accuracy of the continuous method, one can refer to Funke et al. [38] for a thorough examination, in which the authors find that the continuous method has high fidelity to the findings of the discrete method. Taking the same parameters used in this continuous

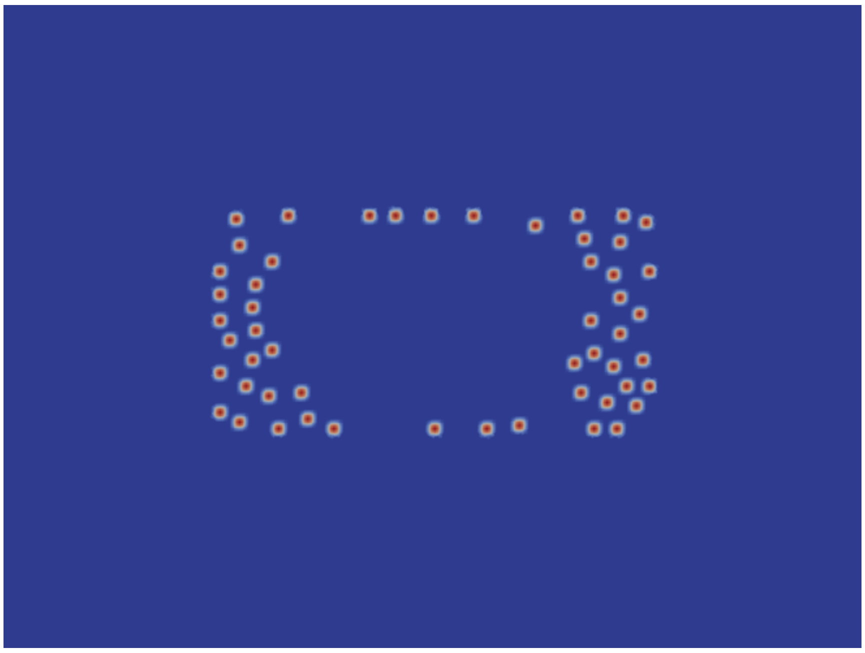


Figure 5.12: Optimal discrete turbine field (using algorithm 12)

optimization, a visual confirmation is easily obtained by optimizing the placement of $N = 52$ turbines using the discrete method in the same domain.

5.9 TSTO Formulated as a Two-Step MIPDECO

Using the continuous model in the previous section as a baseline, this section will present a modified version of the continuous model, whose output (the optimal density field of tidal turbines) serves as the input to a modified version of the MIPDECO formulation presented earlier. The profit equation is modified to reflect time-discounting and and an explicit currency cost of power generation based on the levelized cost of energy (LCOE) methodology and an revenue function based upon UK government subsidies to the tidal stream turbine industry; consequently, no profit margin is assumed.

5.9.1 Step 1: A New Continuous TSTO Model

In the first step, we introduce an explicit quantification of the cost function modelled by the LCOE, for which estimates exist in [47] [48]. The major components of LCOE are the present value of energy output and the present value of energy generation cost, both of which are modulated by a discount rate, which reflects human preference for present satisfaction over future satiation, as well as the private cost of capital.

5.9.2 Discount Rates

Numerous psychological and economic studies have shown that discount rates are complex; generally, these studies revolve around a choice presented to an individual of accepting a reward immediately, or a greater reward in the next time period. For instance, a subject may be offered \$100 today, or \$105 tomorrow. If presented with a selection of similar scenarios, one should theoretically be able to derive a discount rate which measures how an individual values presently-held money and future cash income as a function of time. Though a simple exponential discount rate is often used (i.e. a human equally prefers \$100 immediately to $\frac{100}{(1+r)^t}$ in period t given their subjective discount rate r), researchers such as Kahneman and Tversky (prospect theory), Bordalo et al. (salience theory) have shown that human preferences are intertwined with risk and uncertainty, and are generally quite complex and evidentiary of human irrationality. Studies have revealed well-known situations such as the Allais Paradox, which empirically invalidates expected utility theory, a cornerstone of Economics. Put simply, there is no empirically satisfying theory for universal human discounting.

Discount rates can also be disentangled from single subjects and applied to markets. For instance, when investors are presented with a host of companies seeking investment, discount rates for firms are often tied to the interest rate, industry indicators, and particular aspects of the firm itself. All of these factors come together in a metric known as the cost of capital – i.e. the interest rate which a firm pays on the loans required to fund its projects. Risker projects (i.e. unproven alternative energy projects such as tidal stream turbines) typically incur a higher interest rate than less risky projects, as the implicit risks of default or less-than-optimal returns are priced in by investors who are loaning companies money [47]. Consequently, following Allan et al., a high discount rate of 10% will be used. It would be interesting, however, to survey literature and business decisions in order to find an accurate discount rate for the tidal stream industry in the UK, perhaps incorporating hyperbolic discounting or some other recent empirical findings regarding human discounting and preferences.

5.9.3 Levelized Cost of Energy

Using the LCOE, we can model the lifetime cost of a single turbine with discount rate $r = 10\%$, time period $(0, T)$, value of energy output O_t, and cost C_t at time t as:

$$LCOE = \frac{\sum_{t=0}^{T} C_t/(1+r_t)^t}{\sum_{t=0}^{T} O_t/(1+r_t)^t} \tag{5.26}$$

From this, given a predfined constant $LCOE$, we can easily calculate:

$$\text{Total Discounted Cost} = \sum_{t=0}^{T} C_t/(1+r_t)^t = LCOE \cdot O_t/(1+r_t)^t \tag{5.27}$$

The calculation of energy output has been presented in the previous section. The calculation of cost is out of the scope of this project, though one can refer to Allan et al. [47] for a thorough analysis of the levelized costs of wave and tidal stream energy generation in the United Kingdom. Given the 10% discount rate, Allan et al. found the LCOE for tidal stream energy generation in the UK to be £81.25 MWh in 2006 prices. Adjusting this LCOE for inflation to 2016 prices, I will use an LCOE of £107.89 MWh. Additionally, all further monetary values will be quoted in 2016 prices.

For a single turbine, O_t is simply the energy produced at time t, e_t. I will consider a constant tidal cycle with peak velocity flow $\boldsymbol{u}_c$. As energy generation is considered stationary, the power production at any interval t for a single turbine can be derived from Equation 5.20, where the final answer results from the fact that $\int_\Omega d(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1$ when considering a single turbine.

$$e = \int_\Omega d(\boldsymbol{x}) \cdot \frac{1}{2}\rho C_T A_T \boldsymbol{u}_c^3 \mathrm{d}\boldsymbol{x} = \frac{\rho}{2} C_T A_T \boldsymbol{u}_c^3 \tag{5.28}$$

It is important to note that power is originally measured in watts. I will be using a 5 year testing period, with yearly time steps to reflect time discounting. As LCOE is measured in £/MWh, the energy for each time period e_t is given in MWh by the formula $e_t = \frac{365 \cdot 24 \cdot e}{1e6}$. We then model the total discounted cost per turbine as:

$$C = \frac{\rho}{2} \sum_{t=0}^{T} \frac{LCOE \cdot e_t}{(1 + r_t)^t} \tag{5.29}$$

It is important to note that this cost coefficient assumes that the LCOE is constant each year, and thus changes *only* due to inflation each year, which is controlled for when prices are converted to 2016 GBP.

5.9.4 A Revenue Function

Modelling revenue is a bit trickier than cost as it cannot be modelled per turbine, due to the fact that power extraction (and thus revenue) is highly dependent on configuration of the entire turbine field. Consequently, I will create a revenue function primarily dependent on the 10% discount rate, the power produced by the entire field at each time step, a constant k which corrects for efficiency losses, and the income per unit of energy $I(t)$ at time t.

As Funke et al. [38] note, the energy-capture model is over-optimistic due to neglection of energy mixing and efficiency losses. To penalize the energy optimism, I will set $k = .5$, so that only 50% of theoretically extractable energy is actually captured by the farm. Furthermore, the income per energy unit $I(t)$, is based upon the UK government's Electricity Market Reform (EMR) programme which encourages (and subsidies) renewable electricity generation. The EMR programme includes the Contract for Difference (CfD) mechanism; therein, contracts are set up between the government and private power providers to which the government agrees to the difference between a 'Strike Price' based upon the electricity provider's industry (coal, nuclear, wind, tidal, etc...) and the prevailing market price for each MWh of electricity generated. As the EMR and CfD encourage renewables, the Strike Price in 2012 pounds for tidal stream energy generation is £305/MWh, or £330.51/MWh in 2016 prices (though it is noted that this subsidy is only intended for the first 30MW capacity of any tidal stream project) [49], nearly five times the prevailing power rate. These contracts hold from 2014-2019 (at the same prices), at which point the programme will likely be renewed. Given this information, I will hold the income per energy unit to be constant (growing at the same rate of inflation as the LCOE) so that $I(t) = I =$ £330.51/MWh, though this will be discounted at the aforementioned discount rate. It is important to note that due to the fact that the strike price for tidal stream energy generation is constant through 2019 (other types of energy generation have strike prices which vary from year to year), the income per energy unit is thus temporally constant when we control for inflation and convert the strike price to 2016 dollars. In general, however, this is not true, and the revenue (or cost) function can easily be changed to reflect differences resulting from a LCOE and $I(t)$ which do not grow conterminously with inflation, or predefined yearly variations in the strike price which are codified in the original 2014 ERm CfD contract.

Given these definitions, the total revenue for a *field* for turbines may be modelled as:

$$\text{Total Discounted Revenue} = R(\boldsymbol{u}, d) = k \sum_{t=0}^{T} \frac{I E_t(\boldsymbol{u}, d)}{(1 + r_t)^t} \tag{5.30}$$

Where E_t is the power generated in MWh over the course of each time step and is given by:

$$E_t(\boldsymbol{u}, d) = 365 \cdot 24 \cdot 1e6 \cdot \rho \int_{\Omega} c_t(d(\boldsymbol{x})) ||\boldsymbol{u}(\boldsymbol{x}, t)||_2^3 \mathrm{d}\boldsymbol{x} \tag{5.31}$$

It is also important to note that this form is tailored for the continuous case, in which turbine-induced friction c_t is a function of turbine density $d(\boldsymbol{x})$, though this need not be the case. For instance, friction could be a function of location as in the discrete model, or of the binary variables which activate/deactivate turbines as in the MIPDECO formulation in Equation 5.13.

After implementing time-discounting and an explicit cost and revenue per unit of energy, the first-step continuous model becomes:

$$\max_{d}\ R(\boldsymbol{u}, d) - C\int_{\Omega} d(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \quad s.t. \quad \begin{cases} \frac{\delta \boldsymbol{u}}{\delta t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} - \nu\nabla^2\boldsymbol{u} + g\nabla\eta + \frac{c_b + c_t(d)}{H}||\boldsymbol{u}||\boldsymbol{u} = 0 \\ \frac{\delta\eta}{dt} + \nabla\cdot(H\boldsymbol{u}) = 0 \\ 0 \le d(\boldsymbol{x}) \le \bar{d}(\boldsymbol{x}) \quad \forall \boldsymbol{x}\in\Omega_{farm} \\ d(\boldsymbol{x}) = 0 \quad \forall\boldsymbol{x}\in\Omega\setminus\Omega_{farm} \end{cases} \tag{5.32}$$

Assuming identical velocity over time, this model can easily be solved in the same manner as Funke et al's continuous model, using a gradient-based optimization algorithm. Having used d as the control, the optimal density field is then extracted and used as input to the second step of the algorithm.

5.9.5 Step 2: A New MIPDECO TSTO Model

Using the new continuous model, a density field can be derived which reflects the cost of energy more accurately than when using a model which assumes a certain profit margin. In order to provide a more accurate solution, however, I propose a second step, in which the regions of the density field for which $d^*(\boldsymbol{x}) > 0$ (i.e. regions in which the optimal turbine density is nonzero) are assigned binary integer variables and discrete turbine bumps. Ideally, one could assign every mesh vertex with a nonzero d^* value a binary variable; in practice, while this would massively reduce the required control variables from the mesh dependent MIPDECO case as described in section 5.7, there still may be a prohibitively large number of control variables to allow for optimization. Instead, we may augment Funke et. al's original continuous-to-discrete conversion algorithm by picking a number z (such that $zN > N$ and zN is an integer), and placing zN turbines as opposed to placing an optimal N turbines.

Algorithm 13 Modified Continuous to Discrete Turbine Algorithm

```
1: Determine the optimal number of turbines, $N = \int_\Omega d^*(\boldsymbol{x})\mathrm{d}x$ and choose integer $z$
2: **while** number of turbines deployed is less than $zN$ **do**
3:     Pick a random point $\boldsymbol{x}$ in the domain.
4:     Place a turbine at $\boldsymbol{x}$ with probability $d(\boldsymbol{x})/\bar{d}$ if the minimum distance to all other turbines
       is at least $D_{min}$.
5: Return optimal discrete field of turbines
```

Each of the placed turbines is then assigned a binary variable; finally, with a manageable number of controls, we can solve a MIPDECO which activates up to zN of the binary variables to provide the optimal configuration. This formulation conveniently eliminates the necessity of prespecifying the number of turbines to place (we only need an upper bound), as well as the need for constrained optimization, as algorithm 13 ensures that the minimum distance constraint is satisfied. Consequently, this algorithm may be seen as combining the best of both worlds in a reduced domain – the two step MIPDECO approach is able to optimize *profit* (not energy) over a flexible number of turbines like the continuous approach, but still resolves the turbines individually and micro-sites them as in the discrete approach (though it is noted that the continuous solution is a necessary pre-requisite to the MIPDECO problem), while incorporating useful information gleaned from the initial continuous optimization.

In order to solve the problem, the we must introduce the zN binary variables $w_1...w_{zN}$, each of which correspond to a turbine location as chosen by algorithm 1. In other words, a turbine friction function as in Equation 5.11 is added for each of the zN potential turbine locations. The energy function is changed slightly to be a function of the binary variables $\boldsymbol{w}$ rather than the actual locations they correspond to:

$$E_t(\boldsymbol{u}, \boldsymbol{w}) = \rho \int_\Omega c_t(\boldsymbol{w}) ||\boldsymbol{u}(\boldsymbol{x}, t)||_2^3 \mathrm{d}\boldsymbol{x} \tag{5.33}$$

The power function is changed to reflect its dependence on $\boldsymbol{w}$ rather than the density field $d(\boldsymbol{x})$:

$$\text{Total Discounted Revenue} = R(\boldsymbol{u}, \boldsymbol{w}) = k \sum_{t=0}^{T} \frac{IE_t(\boldsymbol{u}, \boldsymbol{w})}{(1 + r_t)^t} \tag{5.34}$$

Given these changes, the objective functional is slightly modified from the continuous version, as MIPDECO formulation optimizes over binary variables which either activate or deactivate the potentially optimal pre-chosen turbine locations.

$$\max_w \; J(\boldsymbol{w}) = R(\boldsymbol{u}, \boldsymbol{w}) - C \sum_{i=1}^{zN} w_i \tag{5.35}$$

Finally, the second stage MIPDECO is solved:

$$P(\boldsymbol{w}^*) \equiv \max_w \; J(\boldsymbol{w}) \quad s.t. \quad \begin{cases} \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + g \nabla \eta + \frac{c_b + c_t(\boldsymbol{w})}{H} ||\boldsymbol{u}|| \boldsymbol{u} = 0 \\ \nabla \cdot (H\boldsymbol{u}) = 0 \\ w_i \in \{0, 1\} \;\; \forall i \in 1...zN \end{cases} \tag{5.36}$$

Hearkening back to the Source Inversion problem, we arrive at the binary variable issue again. While a branch and bound or branch and cut approach is possible, it is impractical due to the enormity of the mesh ($n + m > 90,000$ in my test case), and thus I will use simple rounding similar to the heuristic integer algorithm used for the Source Inversion problem. First, the binary variables will be relaxed so that $0 \leq w_i \leq 1 \; \forall i \in 1...zN$. This optimization problem can then be solved using L-BFGS-B with bounds on the binary variables. The solution $\boldsymbol{w}^*$ will be an array of continuous values corresponding the intensity of turbine activation (more than half of these binary variables naturally tended to 0 or 1 in my tests). A rounding heuristic can then be used to choose the best N turbine locations.

5.10 Tested Algorithms

In order to compare the various algorithms from Funke et al.'s papers and my own MIPDECO formulation, I tested each algorithm using the same parameters with the goal of maximizing profit.

Algorithm	Description
Continuous Algorithm	Calculates the optimal continuous density field, profit (and power extraction/cost), and number of turbines
Two-Step Algorithm	Uses continuous solution and algorithm 12 to set up an initial layout for the discrete problem with the number of turbines prespecified by the optimal continuous density field
MIPDECO	Uses continuous solution and algorithm 13 as an initial layout for a MIPDECO problem *without* prespecified number of turbines

In each of the three algorithms, the following parameters were used. Note that the velocity of flow is constant and time-invariant, and thus time only impacts the optimization problem through its effect on monetary discounting in the revenue and cost functions. For a justification of the thrust coefficient and turbine blade radius parameters, see Funke et al. [38].

Parameter	Description
Domain Ω	$2400m$ x $2400m$
Farm area Θ	$800m$ x $800m$
Water density ρ	$1000kg/m^3$
Kinematic Viscosity ν	$2m/s$
Gravity g	$9.81m/s^2$
Constant bottom friction c_b	0.0025
Turbine blade radius r_b	$8.35m$
Thrust coefficient C_T	$0.86m$
Minimum distance between turbines	$40m$
Blade-swept surface area A_T	$201.06m^2$
Resting water depth H	$50m$
Velocity of flow $\boldsymbol{u}_{in}$	$2m/s$
Discount rate r_t	0.1
Timeframe t	$(0, 5)$ [5 years, annual steps]
LCOE	£107.89/MWh
Energy revenue I	£330.51/MWh
Efficiency rate k	0.5
Discounted lifetime turbine cost C	£2,971,450.19

Figure 5.13: Tidal stream turbine optimization problem parameters

In order to properly convert units, I assumed that each time step was a year long (primarily for time-discounting purposes). Based upon this, energy output and cost were converted from watts to megawatt hours (using the LCOE and I which are measured in megawatt hours), so that the profit functional and cost are able to be measured in 2016 Great Britain Pound Sterling.

5.11 TSTO Optimization Results

Using the parameters as described in the previous section, as well as the new formulation of the profit objective functional, I obtained the following results from each of the three optimization algorithms, as well as a baseline measurement for the initial two-step layout resulting from algorithm 12. A mesh size of 4m in the farm area and 100m outside the farm area was used, yielding a computational mesh of 90,500 elements. As one can see, the results lie on a spectrum.

Algorithm	Profit (GBP)	Number of Turbines	Runtime (seconds)
Continuous	9.57e7	122.5	7978
Two-Step Discrete	5.04e7	123	38363
MIPDECO	4.021e7	123	9622
Initial Two-Step	3.11e7	123	7979

Figure 5.14: Results for tidal stream turbine optimization (all algorithms)

The continuous formulation automatically yields the optimal number of turbines and supposedly yields the highest profit, though as it does not individually resolve turbines, the profit comparison between the continuous and discrete situations is spurious. Crucially, however, optimization of the continuous case takes a relatively short period of time ($\sim$ 133 minutes), despite the large number of elements in the mesh. After running the continuous example, I placed 123 turbines using algorithm 12. This initial placement from the two-step algorithm yielded 3.11 x 10^7 GBP of profit.

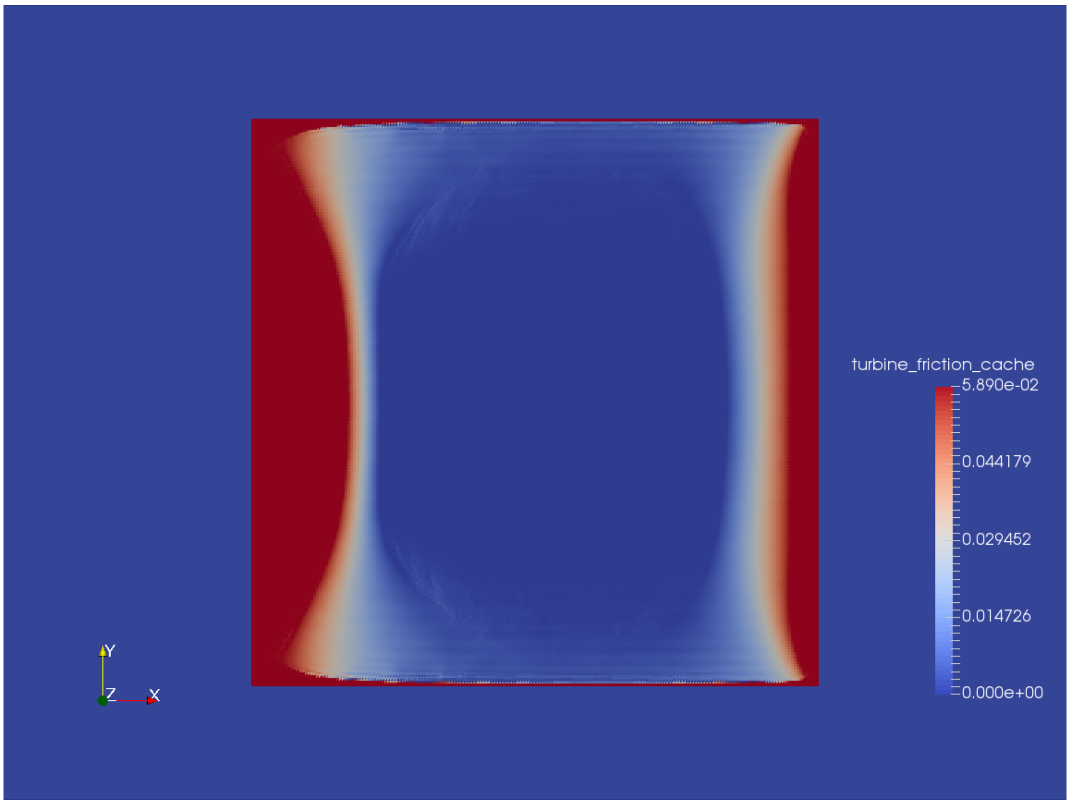


Figure 5.15: Optimal continuous turbine field (domain and farm area shown)

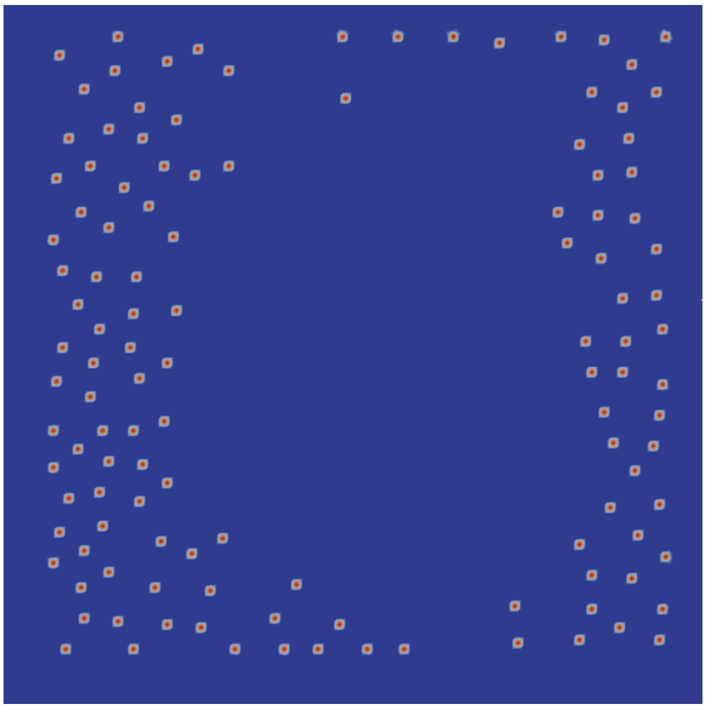


Figure 5.16: Initial two-step turbine field (only farm area shown)

In order to improve the baseline result, I implemented the second step of the two-step algorithm, in which I ran a full discrete optimization using the layout as provided by algorithm 12 as the initial layout. While the optimization resulted in 5.04 x 10^7 GBP of profit, a large increase over the baseline, the increase in profit came at the cost of a severe increase in time complexity as the second step of the two-step algorithm individually resolves the location of each turbine.

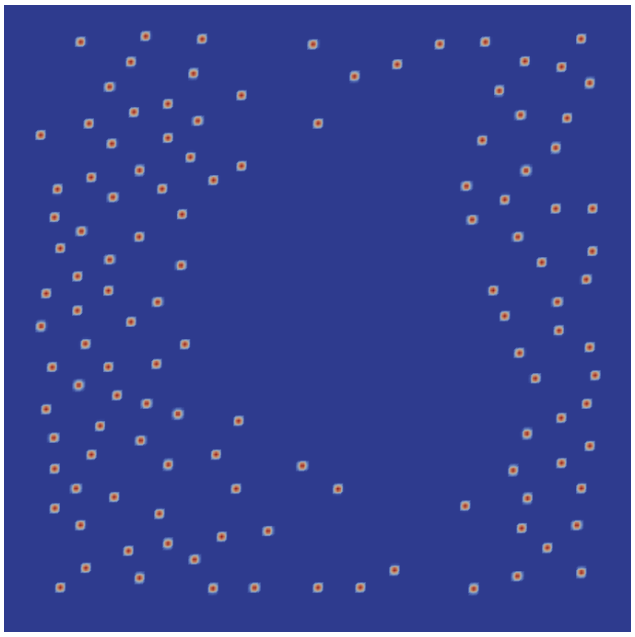


Figure 5.17: Two-step discrete turbine field (only farm area shown)

In order to use the MIPDECO algorithm, I ran algorithm 13 to place 200 turbines using the output of the continuous algorithm. Instead of incrementally modifying the fixed number of turbine locations, the MIPDECO algorithm fixes the possible turbine locations to a subset of the 200 locations, and then optimizes binary variables assigned to each fixed turbine. Consequently, (remembering that the MIPDECO algorithm runtime includes that of the Continuous algorithm) the total MIPDECO runtime is roughly one fifth of the Discrete algorithm's runtime – in this case 2.65 hours instead of 12.87 hours. Additionally, the MIPDECO turbine layout improved profit as compared with the initial two-step heuristic layout by 33.5%, though still yielded far less profit than the discrete case, which improved upon the initial layout by 62.1%. In short, the two-step discrete method returned a a significantly higher profit than the baseline, at the expense of a significant increase in algorithm runtime. The MIPDECO formulation improved moderately upon the baseline, with comparatively little time increase.

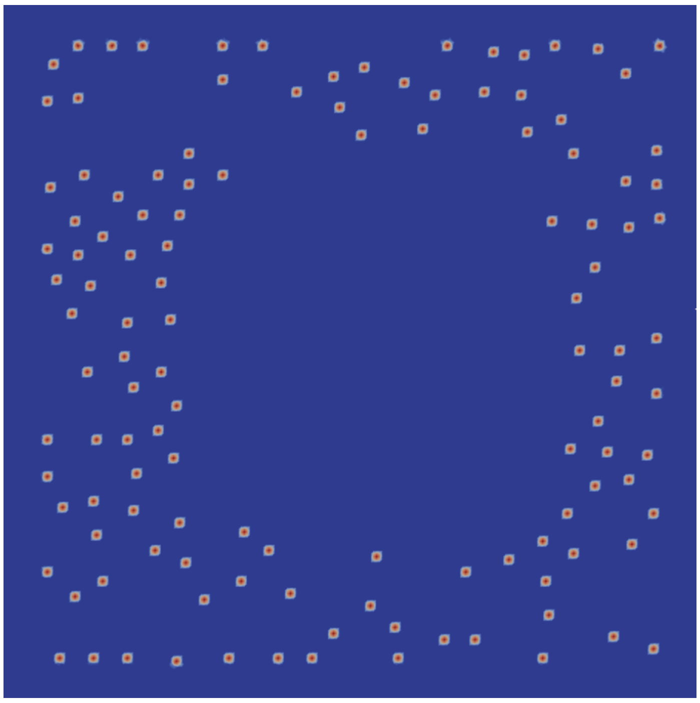


Figure 5.18: Optimal MIPDECO turbine field (only farm area shown)

Chapter 6

Summary and Further Research

This MSc individual project has endeavoured to provide a background to Mixed-Integer PDE-Constrained Optimization (MIPDECO) and introduce gradient-based approximation methods for manipulating and solving the Source Inversion problem and Tidal Stream Turbine Optimization (TSTO) problem in a MIPDECO framework.

6.1 The Source Inversion Problem

In the Source Inversion problem, this project introduced the heuristic integer, penalty, and two-step algorithms in order to solve a MIPDECO with an integer or near-integer solution while using gradient-based methods. These algorithms were benchmarked against a continuous algorithm and branch and bound to compare both accuracy and time complexity. The most important results from this benchmarking are as follows:

1. **The baseline continuous algorithm was implemented in two different ways, which resulted in a trade-off between accuracy and time-complexity.** The continuous algorithm (w control) assigned a single control variable to each source function's binary variable. Optimization runtime for this algorithm increased in the number of source functions (i.e. in the number of controls), but maintained good accuracy across all test settings. The continuous algorithm (sum control) assigned a single control variable to the entire source field (the sum of source functions), resulting in a slower rate of optimization runtime increase in source function number. As a tradeoff, however, the continuous algorithm (sum control) requires a chained two-step optimization problem, with a convex second problem guaranteeing a global optimum *given the globally optimal output of the first stage of the optimization problem.* The inability to guarantee the optimal output from the first stage of the problem resulted in a generally non-robust optimization algorithm with a wildly varying accuracy. Increasing the convergence tolerance on the first stage o the optimization problem may have increased the robustness of the sum control algorithm, though at the expense of greatly increasing algorithm run-time which already bottlenecked at the first stage.

2. **The penalty and two-step algorithms were not robust.** Both algorithms failed to provide a consistent integer solution due to difficulties in calculating the α hyperparameter, which in many test situations still required tuning at 7 significant figures. If one is able to calculate the proper α, however, these algorithms hold great promise.

3. **The heuristic integer algorithm performed quite well against the penalty/two-step algorithms.** In many of the test scenarios, rounding the largest optimal continuous values of the decision variables to 1 and all else to 0 mostly coincided with the exact integer

solution obtained from branch and bound. Further research is needed to determine whether this correspondence holds true in a wide variety of problems with different parameter configurations, or if the excellent heuristic integer algorithm results are due to the specific nature and parameters of this problem.

4. **Branch and bound was the best algorithm when the number of total variables was small, but timed out with a large number of total variables.** When the number of total variables (source function number n^2 + number of mesh vertices m^2) was small, branch and bound returned a configuration of activated source functions yielding the least – and assuredly global minimum – reconstruction error, while also taking the least amount of time. Yet the branch and bound solver's time complexity increased exponentially with a large coefficient, while the gradient-based algorithms were subject to exponential time complexity with a lower coefficient (i.e. with $K = n^2 + m^2$, branch and bound was $\mathcal{O}(a^K)$ while gradient algorithms were $\mathcal{O}(b^K)$, where $a >>> b$) and thus a drastically smaller runtime. In essence, it appears that neither MINLP or gradient-based methods are strictly better for the Source Inversion problem, though each is uniquely suited to any given situation depending on the refinement of the mesh and the number of decision variables.

5. **Future research could examine the effect of varying the Gaussian hyperparameters or changing the nature of the source functions to a non-Gaussian type.** While I tested a slew of parameter configurations for the Source Inversion problems, more hyperparameter diversity is required in order to extrapolate broader conclusions. For instance, it is difficult to assess whether the two-step/penalty methods will be useful without examining the difficulty of finding α in a host of different situations. Furthermore, the correspondence between branch and bound results and the heuristic integer algorithm needs to observed in a broader variety of test cases before a more robust linkage can be proposed.

6.2 Tidal Stream Turbine Optimization

In Chapter 5, this project attempted to approach TSTO from a MIPDECO standpoint, fusing the use of binary variables for individual turbines (bump functions) with gradient-based discrete and continuous optimization methods as developed in a series of papers by Funke et al. [38] [40] [41]. The most important contributions and results from this case study are summarized below.

1. **A new profit-based functional was defined for the TSTO problem, which takes revenue, cost, and time-discounting into account.** I used Levelized Cost of Energy estimates for the UK energy sector and defined a discount rate from data in Allan et al. [47] and Astariz et al. [48], as well as government subsidies for TST energy [49] to create a profit-based (as opposed to power-based) functional. This functional does not rely on a prespecified profit margin and can be modified easily to include differing rates of revenue/cost inflation or depreciation.

2. **A great share of runtime differences in TSTO benchmarking can be traced to the requirements (or lack thereof) of minimum distance bounds.** The discrete model attempted to micro-site turbines by using discrete bumps which are subject to inequality constraints and are able to move freely throughout the domain. This increased flexibility resulted in the highest objective function value (the most power/profit extracted) at the expense of requiring a pre-specified number of turbines, and poor scalability properties due to its high time complexity. As opposed to the time-complex and

powerful discrete model, the continuous model included no constraints or individually resolved turbines, but rather assigned a turbine density to each region of the discretized domain. Consequently, the continuous algorithm loses quite a bit of accuracy, but is an extremely efficient way to roughly estimate an optimized turbine farm as well as to provide starting locations for the discrete algorithm. The initial two-step algorithm chained the continuous and discrete model together, converting the optimal continuous density field to a discrete turbine farm by using algorithm 12 to stochastically place individual turbines with probability of placing a turbine on a mesh region increasing in its assigned optimal turbine density. The full two-step algorithm optimized the initial two-step algorithm's turbine farm (as constructed by algorithm 12) using the discrete method, once again using the flexible, but time-demanding minimum distance constraints. Finally, the MIPDECO algorithm took an initial layout of turbines as determined by algorithm 13; no minimum distance constraints were introduced, and thus possible turbine locations of the MIPDECO algorithm were restricted to a preselected number of mesh regions (greater than the optimal number of turbines as determined by the continuous model) selected with probability increasing in optimal turbine density.

3. **The MIPDECO algorithm poached the best aspects of the discrete, continuous and full two-step models by creating a time-efficient model which does not require a prespecified number of turbines and retains most of the accuracy of the discrete model.** The key aspect of the MIPDECO model is to use algorithm 13 to place a superset of potential turbines. Assuming that the continuous algorithm determines the optimal number of turbines to be N, algorithm 13 places $z{\cdot}N$ initial turbines, where $z > 1$. With a manageable number of binary variables, the MIPDECO algorithm maximizes profit, automatically finding the number of optimal turbines (guided by, but independent of the continuous model's determination), while maintaining low time complexity, and retaining most of the power of the discrete model. While providing functionality for MIPDECO optimization with inequality constraints (i.e. using SQP instead of L-BFGS-B) and allowing for initial turbine placements to violate minimum distance constraints would improve the final profit yielded (as well as allow for far more initial turbines to be placed), this increase in flexibility would come at the cost of severely increasing runtime, possibly removing all of the time advantages of the MIPDECO algorithm.

4. **Against the discrete, continuous, and two-step algorithms, the MIPDECO algorithm provided a jack-of-all-trades result.** The MIPDECO algorithm yielded a significantly higher profit than the initial two-step algorithm while requiring a negligible increase in time, but significantly less profit than the full two-step algorithm which requires a significantly longer runtime due to the need for each turbine to be individually resolved. This was the expected result, as the MIPDECO algorithm optimizes binary variables for $z \cdot N$ locations – and thus is more flexible than the initial two-step algorithm which simply places N turbines sans any further optimization)– but does not add inequality constraints (and so is more restrictive than the discrete or full two-step algorithms, which are not restricted to $z \cdot N$ predetermined locations).

5. **This work has demonstrated that embedding a MIPDECO optimization problem as a second step in a continuous TSTO problem may prove fruitful if minimizing algorithm runtime is desired.** Further research could seek a way to define a metric with which to measure the optimality of z or to introduce a heuristic which allows z to be tailored to a specific problem. Another useful study might extend the MIPDECO algorithm to incorporate the minimum-distance constraint and use SQP for optimization.

Additionally, the development of novel continuous-to-integer conversion heuristics would likely prove quite useful when optimizing the field of binary variables using gradient-based methods.

Bibliography

[1] Magagna, D., and Uihlein, A., “Ocean energy development in Europe: Current status and future perspectives,” International Journal of Marine Energy, vol. 11, pp. 84–104, 2015. pages 1

[2] Wave and Tidal Energy Strategic Technology Agenda (Rep.). (2014). Strategic Initiative for Ocean Energy. https://www.oceanenergy-europe.eu/images/projects/SI-Ocean—WaveTidal-Strategic-Technology-Agenda.pdf pages 1

[3] Bretscher, O. (2014). Linear algebra with applications. Pearson. pages 7

[4] Peterson, K. B. & Pederson, M. S. (2012, November). The Matrix Cookbook [PDF]. pages 7

[5] Chong, E. K. P. and Zak, S. H., An Introduction to Optimization. Somerset: Wiley, 2011. pages 7, 9, 11, 12, 86

[6] Evans, L. C. (1998). Partial differential equations (2nd ed.). Providence: American Mathematical Society. pages 17

[7] Kuhn, H. W. and Tucker, A. W. Nonlinear programming. Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1950, pp. 481–492. University of California Press, Berkeley and Los Angeles, 1951. pages 9

[8] Karush, W. Minima of functions of several variables with inequalities as side constraints. Master’s thesis, University of Chicago, Chicago, IL, USA, 1939. pages 9

[9] Gomory, Ralph Edward. An algorithm for the mixed integer problem.. Santa Monica, CA: RAND Corporation, 1960. http://www.rand.org/pubs/research_memoranda/RM2597.html. Also available in print form. pages 10

[10] Land, A. H., and Doig, A. G., “An Automatic Method of Solving Discrete Programming Problems,” Econometrica, vol. 28, no. 3, p. 497, 1960. pages 10

[11] Dakin, R. J.. “A tree-search algorithm for mixed integer programming problems,” The Computer Journal, vol. 8, no. 3, pp. 250–255, Jan. 1965. pages 10

[12] Cornuéjols, G. C. A.. “Revival of the Gomory cuts in the 1990’s,” Annals of Operations Research, vol. 149, no. 1, pp. 63–66, Feb. 2006. pages 10

[13] Padberg, M., and Rinaldi, G., “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems,” SIAM Review, vol. 33, no. 1, pp. 60–100, 1991. pages 10

[14] Achterberg, T., Berthold, T., Koch, T., and Wolter, K., “Constraint Integer Programming: A New Approach to Integrate CP and MIP,” Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems Lecture Notes in Computer Science, pp. 6–20. pages

[15] Elman, H. C., Silvester, D. J., and Wathen, A. J., Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Oxford: Oxford University Press, 2005. pages

[16] Oliver, Y. P. Z., “Comparison of finite difference and finite volume methods the development of an education tool for the fixed-bed gas absorption problem,” thesis, 2011. National University of Singapore, Department of Chemical and Biomolecular Engineering. pages

[17] Botte, G. G., Ritter, J. A., and White, R. E., “Comparison of finite difference and control volume methods for solving differential equations,” Computers Chemical Engineering, vol. 24, no. 12, pp. 2633–2654, 2000. pages 19

[18] Peiro, J., and Sherwin, S., ”FINITE DIFFERENCE, FINITE ELEMENT AND FINITE VOLUME METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS” In Handbook of Materials Modeling. Volume I: Methods and Models (2005), pp. 1-32 edited by S. Yip pages 19

[19] ESI-Group: The Virtual Space Try Out Company. “FEM-FDM”. [powerpoint slides] Available at: http://www.delcamural.ru/files/R007pages

[20] Haber, E. ”Mini-Course on PDE-Constrained Optimization” University of British Columbia, Canada, 2011. pages 22

[21] M. Heinkenschloss, ”PDE-Constrained Optimization”, Siam Conference on Optimization, Boston, Massachusetts, 2008. pages 22

[22] Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A. (2013). Mixed-integer nonlinear optimization. Acta Numerica, 22, 1-131. doi:10.1017/s0962492913000032 pages 15

[23] Cay, P., Leyffer, S., van Bloemen Waanders, B., Kouri, D., Thuenen, A. SAMSI Optimization Workshop. Research Triangle Park. pages 24

[24] Funke SW. The automation of PDE-constrained optimisation and its applica- tions [Ph.D. thesis]. London, UK: Imperial College London; 2012. pages 26

[25] S. Sager, H. G. Bock, and G. Reinelt, “Direct methods with maximal lower bound for mixed-integer optimal control problems,” Mathematical Programming, vol. 118, no. 1, pp. 109–149, 2007. pages 25, 35

[26] Huang, C.Y. and Wang, T.C., GRADIENT METHODS FOR BINARY INTEGER PROGRAMMING. pages 30

[27] Tidal Energy. (n.d.). Retrieved April 14, 2017, from http://www.oceanenergycouncil.com/ocean-energy/tidal-energy/ pages 52

[28] Tidal giants - the world’s five biggest tidal power plants. (n.d.). Retrieved April 14, 2017, from http://www.power-technology.com/features/featuretidal-giants—the-worlds-five-biggest-tidal-power-plants-4211218/ pages 52

[29] Tidal Barrage and Tidal Barrage Energy Devices. (n.d.). Retrieved April 14, 2017, from http://www.alternative-energy-tutorials.com/tidal-energy/tidal-barrage.html pages iv, 53

[30] Charlier, R. (2007). Forty candles for the Rance River TPP tides provide renewable and sustainable power generation. Renewable and Sustainable Energy Reviews, 11(9), 2032-2057. doi:10.1016/j.rser.2006.03.015 pages 53

[31] Retiere, C. (1994). Tidal power and the aquatic environment of La Rance. Biological Journal of the Linnean Society, 51(1-2), 25-36. doi:10.1111/j.1095-8312.1994.tb00941.x

[32] Steijn, R. (2015, January 13). Dynamic Tidal Power Technology Advances. Retrieved April 14, 2017, from http://www.renewableenergyworld.com/articles/2015/01/dynamic-tidal-power-technology-advances.html pages 53

[33] Hanania, J., Hefferman, B., Stenhouse, K., & Donev, J. (n.d.). Dynamic tidal power. Retrieved April 14, 2017, from http://energyeducation.ca/encyclopedia/Dynamic_tidal_power pages 54

[34] Swansea Bay. (n.d.). Retrieved April 14, 2017, from http://www.tidallagoonpower.com/projects/swansea-bay/ pages

[35] Tidal energy project to be constructed in the Pentland Firth. (2014, August 22). Retrieved April 14, 2017, from http://www.bbc.co.uk/news/uk-scotland-28887542 pages

[36] MeyGen — Tidal Projects. (n.d.). Retrieved April 14, 2017, from https://www.atlantisresourcesltd.com/projects/meygen/ pages

[37] Tidal Turbines — Installation Procurment. (n.d.). Retrieved April 20, 2017, from https://www.atlantisresourcesltd.com/services/turbines/ pages

[38] Funke, S., Kramer, S., Piggott, M. (2016). Design optimisation and resource assessment for tidal-stream renewable energy farms using a new continuous turbine approach. Renewable Energy, 99, 1046–1061. doi:10.1016/j.renene.2016.07.039 pages 56, 61

[39] Culley, D. M., Kramer, S. C., Funke, S. W., & Piggott, M. D. (2014). A Hierarchy of Approaches for the Optimal Design of Tidal Turbine Arrays. ICOE 2014 (5th International Conference on Ocean Energy). pages 56, 57, 58, 65, 67, 68, 71, 74, 79

[40] S. Funke, P. Farrell, M. Piggott, ”Tidal turbine array optimisation using the adjoint approach. Renewable Energy, vol. 63, pp. 658–673, 2014. doi:10.1016/j.renene.2013.09.031 pages 57

[41] Culley, D., Funke, S., Kramer, S., Piggott, M. (2016). Integration of cost modelling within the micro-siting design optimisation of tidal turbine arrays. Renewable Energy, 85, 215–227. doi:10.1016/j.renene.2015.06.013 pages 57, 58, 59, 61, 79

[42] G. L. Barnett, S. W. Funke, and M. D. Piggott. Hybrid global-local optimisation algorithms for the layout design of tidal turbine arrays. Submitted to Renewable Energy, 2014. pages 57, 58, 79

[43] P. Y. Zhang, D. A. Romero, J. C. Beck, C. H. Amon, ”Solving wind farm layout optimization with mixed integer programs and constraint programs,” EURO Journal on Computational Optimization, 2(3), pp. 195–219, 2014. doi:10.1007/s13675-014-0024-5 pages 57, 58

[44] P. Fagerfjall, "Optimizing wind farm layout – more bang for the buck using mixed integer linear programming", Master's thesis. Chalmers University of Technology. Gothenburg, Sweden, 2010. pages 58

[45] Randall, D. A. (2006). The Shallow Water Equations. Retrieved April 20, 2017, from http://kiwi.atmos.colostate.edu/group/dave/pdf/ShallowWater.pdf pages 58

[46] Abolghasemi, M. A., Piggott, M. D., Spinneken, J., Viré, A., Cotter, C. J., Crammond, S. (2016). Simulating tidal turbines with multi-scale mesh optimisation techniques. Journal of Fluids and Structures, 66, 69-90. doi:10.1016/j.jfluidstructs.2016.07.007 pages 58

[47] Allan, G. J., Gilmartin, M., McGregor, P. G., Swales, K. (2011). Levelised costs of wave and tidal energy in the UK: cost competitiveness and the impact of 'banded' Renewables Obligation Certificates. Energy Policy, 39(1), 23-39. DOI: 10.1016/j.enpol.2010.08.029 pages 59

[48] Astariz, S., Vazquez, A., Iglesias, G. (2015). Evaluation and comparison of the levelized cost of tidal, wave, and offshore wind energy. Journal of Renewable and Sustainable Energy, 7(5), 053112. doi:10.1063/1.4932154 pages 69, 70, 79

[49] Electricity Market Reform (EMR). (2015, August 26). Retrieved April 25, 2017, from https://www.ofgem.gov.uk/electricity/wholesale-market/market-efficiency-review-and-reform/electricity-market-reform-emr pages 69, 79

[50] M. B. Giles and N. A. Pierce, "An Introduction to the Adjoint Approach of Design," Flow, Turbulence, and Combustion, vol. 65, pp. 393–415, 2000. pages 71, 79

[51] S. W. Funke and P. E. Farrell, "A framework for automated PDE-constrained optimisation", 2013, submitted. arXiv:1302.3894 [cs.MS] pages 89
pages 88, 89

Appendix A

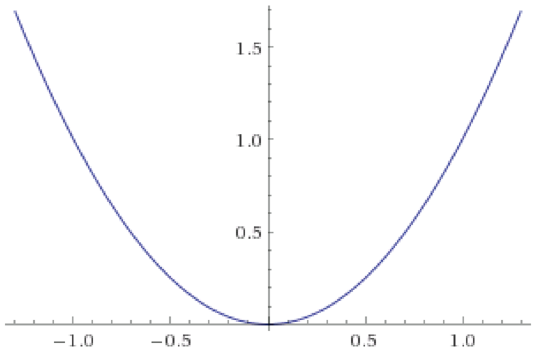
Appendix

A.1 Convexity and MIPDECO

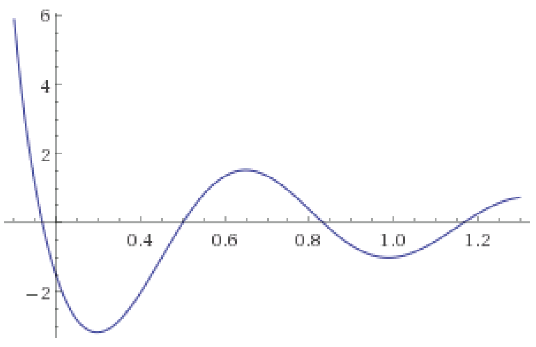
It would be remiss of me not to distinguish between convex and nonconvex optimization. Formally, a function is said to be convex if, for a convex set $\mathcal{X}$ and function $f : \mathcal{X} \to \mathbb{R}$:

$$f(\alpha \boldsymbol{x_1} + (1-\alpha)\boldsymbol{x_2}) \leq \alpha f(\boldsymbol{x_1}) + (1-\alpha) f(\boldsymbol{x_2}) \quad \forall \boldsymbol{x_1}, \boldsymbol{x_2} \in \mathcal{X},\ \forall \alpha \in [0,1] \tag{A.1}$$

Intuitively, the definition in Equation A.1 says that forming a line segment between any two points of the convex function must result in a line which is above the function everywhere between the two chosen points. Below, the graph on the left clearly satisfies this requirement, while the graph on the right does not. This leads to an important result of convexity: any convex function has a single unique global minima, while the same is not true of nonconvex functions.



(a) A convex function with a single local and global minima)



(b) A nonconvex function with a multiple local minima

Figure A.1: Convex (left) and nonconvex (right) functions

Following from the above, any local minimizer of a convex optimization problem will also be a unique global minimizer, while nonconvex functions may have multiple local minimizers [5]. Thus if we solve an optimization problem with decision variable $\boldsymbol{x}$ using an appropriate optimization technique for a minimizer $\boldsymbol{x}^*$:

$$\text{Optimization Problem is Convex: } \boldsymbol{x}^* \implies \boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathbb{R}^n$$

$$\text{Optimization Problem is not Convex: } \boldsymbol{x}^* \not\Longrightarrow \boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathbb{R}^n$$

In other terms, convexity gives a certificate of optimality, while nonconvexity still allows one to find a good point – a point corresponding to one of the two local minimas in Figure 2 – but *not necessarily* the best point.

A.2 Discretizing the Source Inversion Problem

In this optimization problem, there is a reference PDE $\bar{\boldsymbol{u}}$ and a source PDE $\boldsymbol{u}$, both of which are created by finding the solutions to the weak formulation of the Poisson equation, using the reference function or the sum of source functions as the respective source terms. In the general case, we can apply the finite element method to the Poisson equation in the following way (note that a full mathematical treatment of the method is outside of the scope of this individual project):

Suppose we have a Poisson Equation defined in Ω with a Dirichlet boundary condition. We want to find a $\boldsymbol{u}$ such that:

$$\begin{cases} \Delta \boldsymbol{u} = f & \in \Omega \\ \boldsymbol{u} = 0 & \text{on } \delta\Omega \end{cases}$$

Now if $\boldsymbol{u}$ solves the above system, then for any test function v that satisfies the boundary conditions of the above system, the following holds, with $\mathcal{H}$ being the appropriate Sobolev space:

$$\int_\Omega fv \,\mathrm{d}x = \int_\Omega \Delta \boldsymbol{u} v \,\mathrm{d}x \quad \boldsymbol{u}, v \in \mathcal{H} \tag{A.2}$$

Integrating the right hand side by parts (noting that $\Delta\boldsymbol{u}$ is the Laplacian and $\langle ., . \rangle$ is the $L2$ inner product):

$$\int_\Omega fv \,\mathrm{d}x = -\int_\Omega \langle \nabla \boldsymbol{u}, \nabla v \rangle \,\mathrm{d}x \quad \boldsymbol{u}, v \in \mathcal{H} \tag{A.3}$$

The finite element method then finds a suitable $\boldsymbol{u}^d \in \mathcal{V}$, such that Equation A.3 holds for all test functions $v \in \mathcal{V}$, where $\mathcal{V}$ is a finite dimensional subspace of $\mathcal{H}$ (for this problem I use the space of linear piecewise functions). In short, this method allows one to replace a PDE defined in some arbitrary dimension with a tractable approximation to it by using a test function and a subspace of the original domain. Applying the FEM to the present problem, and noting the sign change in the constraint, the finite element method finds a solution vector $\boldsymbol{u}^d$ (which is the solution to the discretized source PDE) such that the following equation holds (with or without the w_{kl} depending on the algorithm used):

$$\int_\Omega \langle \nabla \boldsymbol{u}^d, \nabla v \rangle \mathrm{d}x = \sum_{k,l} (w_{kl} \cdot \int_\Omega f_{kl} \cdot v \ \mathrm{d}x) \quad \forall v \in \mathcal{V} \tag{A.4}$$

In order to create $\bar{\boldsymbol{u}}^d$, the solution to the discretized reference PDE, I assume that the reference function is of the same form as the source functions (where i, j are not on the vertices of the mesh):

$$\bar{f_{ij}} = a \cdot \exp\left(\frac{-(x_i - x)^2 - (y_j - y)^2}{\sigma^2}\right) \quad \in \Omega; \quad i, j \notin \mathbb{V}$$

I use the finite element method to find a $\bar{\boldsymbol{u}}^d$ such that

$$\int_\Omega \langle \nabla \bar{\boldsymbol{u}}^d, \nabla v \rangle \mathrm{d}x = \int_\Omega \bar{f_{ij}} v \ \mathrm{d}x \quad \forall v \in \mathcal{V} \tag{A.5}$$

A.3 Discretizing the TSTO Problem

The discretization process for TSTO generally uses the same finite element method, albeit with a more complex PDE. Choosing the test functions ψ and ϕ, the two shallow water equations are discretized by finding the depth-averaged velocity, and free surface displacement $\boldsymbol{u}^d, \eta^d$, respectively. Thus the goal is to find $\boldsymbol{u}^d, \eta^d \in V$ x W such that $\forall \psi, \phi \in V$ x W, the following equations hold.

$$\left\langle \boldsymbol{u}^d \cdot \nabla \boldsymbol{u}^d, \psi \right\rangle_\Omega + \nu \left\langle \nabla \boldsymbol{u}^d, \nabla \psi \right\rangle_\Omega + g \left\langle \nabla \eta^d, \psi \right\rangle_\Omega + \left\langle \frac{c_b + c_t(m)}{H} ||\boldsymbol{u}^d|| \boldsymbol{u}^d, \psi \right\rangle_\Omega = 0 \tag{A.6}$$

$$- \left\langle H\boldsymbol{u}^d, \nabla \phi \right\rangle_\Omega + \left\langle H\boldsymbol{u}^d \cdot \boldsymbol{n}, \phi \right\rangle_{\delta\Omega_{in} \cup\, \delta\Omega_{out}} = 0 \tag{A.7}$$

As above, $\langle ., . \rangle_\Omega$ denotes the $L2$ inner product over domain Ω, and $\boldsymbol{n}$ is the outward facing normal vector to the boundary of the domain, $\delta\Omega$. Additionally, V is chosen to be the space of all continuous piecewise quadratic functions, while W is the space of all piecewise linear functions.

The above discretization scheme assumes time invariance. That is, we assume the velocity $\boldsymbol{v}$ to enter the farm at a constant rate over the lifetime of the farm. In reality, the farm exists in time period $(0, T)$ with time step δt. Assuming a constant velocity of flow, however, results in the discretization in each time period being exactly the same; thus it is done only once and can be applied across all time steps by scalar multiplication. When the velocity of flow does vary over time – for instance, if we assume a sinusoidal tidal cycle and the aforementioned time domain, then, with k denoting the current time, δt the time step, the following shallow water equations must be solved for each time step during each iteration. It is assumed that initial conditions $\boldsymbol{u}_0, \eta_0$ are supplied.

$$\left\langle \frac{\boldsymbol{u}_k^d - \boldsymbol{u}_{k-1}^d}{\delta t}, \psi \right\rangle_\Omega + \left\langle \boldsymbol{u}_k^d \cdot \nabla \boldsymbol{u}_k^d, \psi \right\rangle_\Omega + \nu \left\langle \nabla \boldsymbol{u}_k^d, \nabla \psi \right\rangle_\Omega + g \left\langle \nabla \eta_k^d, \psi \right\rangle_\Omega + \left\langle \frac{c_b + c_t(m)}{H_k} ||\boldsymbol{u}_k^d|| \boldsymbol{u}_k^d, \psi \right\rangle_\Omega = 0 \tag{A.8}$$

$$\left\langle \frac{\eta_k^d - \eta_{k-1}^d}{\delta t}, \psi \right\rangle_\Omega - \left\langle H_k \boldsymbol{u}_k^d, \nabla \phi \right\rangle_\Omega + \left\langle H_k \boldsymbol{u}_k^d \cdot \boldsymbol{n}, \phi \right\rangle_{\delta\Omega_{in} \cup \delta\Omega_{out}} = 0 \tag{A.9}$$

A.4 The Adjoint Approach to Design

Consider the PDE-constrained optimization problem with PDE $\mathcal{F}$ and decision variable $\boldsymbol{m}$. Note that the objective functional, as well as $\boldsymbol{u}$ are implicitly parameterized by control $\boldsymbol{m}$. Our objective is to minimize the objective functional by picking the optimal controls $\boldsymbol{m}^*$ using a gradient-based method.

$$\min_m \; J(\boldsymbol{u}(\boldsymbol{m}), \boldsymbol{m}) \quad s.t. \quad \begin{cases} \mathcal{F}(\boldsymbol{u}(\boldsymbol{m}), \boldsymbol{m}) = 0 \\ g(\boldsymbol{m}) = 0 \\ h(\boldsymbol{m}) \leq 0 \\ \boldsymbol{m}, \boldsymbol{u} \in \mathcal{H}_1, \mathcal{H}_2 \end{cases} \tag{A.10}$$

As mentioned in Chapter 2, the optimal parameters $\boldsymbol{m}^*$ of PDE-constrained problems are usually found via a first order iterative method such as gradient descent (or second-order approximations using first-order information such as BFGS), as gradient-free approaches are generally generally computationally intractable for any practical applications of PDE-constrained optimization [51]. Then to solve for the gradient of $\hat{J} = J(\boldsymbol{m})$, we can use one of three methods. We could approximate each partial derivative $\frac{d\hat{J}}{\delta m_i}$, $\forall i \in 1, ..., m$, which is generally a poor method due to the difficulties of choosing the perturbation for each decision variable (and there are generally many) [51]. The better options are to either solve forward (i.e. the primal problem), or formulate and solve backward (i.e. the dual, or adjoint problem). The optimal strategy will depend on the number of inputs and outputs. As shown by Giles and Pierce [50] and Funke et al. [51], the adjoint formulation is generally superior to the primal formulation in the PDE-constrained optimization case, due to the presence of a single objective functional and a large number of decision variables. An overview of the mathematical justification for preferring adjoint-based optimization follows.

A.4.1 Solving the Primal Problem

To solve the primal, or forwards optimization problem, the gradient of the objective functional is found using the multivariate chain rule:

$$\frac{d\hat{J}}{d\boldsymbol{m}} = \frac{\delta J}{\delta \boldsymbol{u}} \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{m}} + \frac{\delta J}{\delta \boldsymbol{m}} \tag{A.11}$$

While we have an equation for the objective functional and can generally calculate the direct derivatives $\frac{\delta J}{\delta \boldsymbol{u}}$ and $\frac{\delta J}{\delta \boldsymbol{m}}$, we have no concrete function $\boldsymbol{u(m)}$, which is only implicitly defined by any particular choice of decision variables $\boldsymbol{m}$. Using the chain rule on the original system of PDE constraints, however, one can fix a value for $\boldsymbol{m}$, say $\boldsymbol{m}_t$ (i.e. the value of the parameters at the t'th iteration of gradient descent), and then solve for the total derivative of the PDE solution vector with respect to the decision variables. In other words, we evaluate $\frac{\mathrm{d}(\boldsymbol{u},\boldsymbol{m})}{\mathrm{d}\boldsymbol{m}_t}$ to find the Jacobian $\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{m}_t}$, where $\mathcal{F}$ is from equation A.11. Note that this Jacobian is (assuming $\boldsymbol{u} \in \mathbb{R}^u, \boldsymbol{m} \in \mathbb{R}^m$) of dimension u x m. Assuming that there are many decision variables and many PDE constraints, this matrix will be quite large. Taking our derivative, we end up with the tangent linear system of the PDE constraint:

$$\frac{\delta \mathcal{F}(\boldsymbol{u},\boldsymbol{m})}{\delta \boldsymbol{u}} \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{m}_t} = -\frac{\delta \mathcal{F}(\boldsymbol{u},\boldsymbol{m})}{\delta \boldsymbol{m}_t} \tag{A.12}$$

In this system, $\frac{\delta \mathcal{F}(\boldsymbol{u},\boldsymbol{m})}{\delta \boldsymbol{u}}$ is the linearisation around a certain solution $\boldsymbol{u}$ [51]. This system can be constructed and solved during every iteration of gradient descent, but due to the memory requirements likely will not be possible. If it were, however, one must note that these gradients are with respect to any arbitrary functional $J(\boldsymbol{m})$. Consequently, solving this primal problem is only efficient if the input parameters $\boldsymbol{u}, \boldsymbol{m}$ are small, whereas the number of outputs (functionals) is large. In other words, if we have a few decision variables which exist in a small parameter space but a large number of objective functionals, then the primal approach may be efficient. Generally, however, a PDE constrained optimization problem has a single objective functional that is known before undertaking the optimization process, and a large number of PDE constraints and decision variables. Thus the primal approach is generally inefficient due to the large set of parameters and parameter space [51].

A.4.2 Solving the Dual (Adjoint) Problem

Using the adjoint method, instead of fixing our vector of decision variables $\boldsymbol{m} \in \mathbb{R}^m$, we fix our objective functional $J \in \mathbb{R}$. By assuming that $\frac{\delta \mathcal{F}(\boldsymbol{u},\boldsymbol{m})}{\delta \boldsymbol{u}}$ is invertible, one can derive an expression for the total derivative of the functional with respect to the decision variables, take the Hermetian transpose (just the transpose if all entries of the various Jacobians are real) and solve for the adjoint equation. Thus we have:

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}\boldsymbol{m}_t} = -\left(\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{u}}\right)^{-1}\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{m}_t} \tag{A.13}$$

$$\frac{\mathrm{d}\hat{J}}{\mathrm{d}\boldsymbol{m}_t} = -\frac{\delta J}{\delta\boldsymbol{u}}\left(\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{u}}\right)^{-1}\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{m}_t} + \frac{\delta J}{\delta\boldsymbol{m}_t} \tag{A.14}$$

Taking the Hermetian transpose and creating a new variable $\boldsymbol{\lambda}$:

$$\frac{\mathrm{d}\hat{J}^*}{\mathrm{d}\boldsymbol{m}_t} = -\left(\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{m}_t}\right)^{*}\left(\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{u}}\right)^{-*}\frac{\delta J^*}{\delta\boldsymbol{u}} + \frac{\delta J^*}{\delta\boldsymbol{m}_t} \tag{A.15}$$

$$\boldsymbol{\lambda} = \left(\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{u}}\right)^{-*}\frac{\delta J^*}{\delta\boldsymbol{u}} \tag{A.16}$$

$$\left(\frac{\delta\mathcal{F}\left(\boldsymbol{u},\boldsymbol{m}\right)}{\delta\boldsymbol{u}}\right)^{*}\boldsymbol{\lambda} = \frac{\delta J^*}{\delta\boldsymbol{u}} \tag{A.17}$$

Note that $\boldsymbol{\lambda}$ is the dual (or adjoint) variable of $\boldsymbol{u}$, so that λ_i is the dual variable for u_i. Correspondingly, $\left(\frac{\delta\mathcal{F}(\boldsymbol{u},\boldsymbol{m})}{\delta\boldsymbol{u}}\right)^*$ is the Hermetian (or normal if all entries are real) transpose of the tangent linear operator (i.e. it is the constraint matrix of the dual formulation of this optimization problem). Importantly, the transposition of the primal constraint matrix reverses all information flow so that the the adjoint PDE will run backward in time instead of forward, or the flow of water will be downstream instead of upstream, and so on. Now all of the above information has important ramifications for the problem: like the forward tangent system, $\left(\frac{\delta\mathcal{F}(\boldsymbol{u},\boldsymbol{m})}{\delta\boldsymbol{u}}\right)^*$ is also linear in $\boldsymbol{u}$, and thus linear in the adjoint variable $\boldsymbol{\lambda}$ as well. Then if we solve for $\boldsymbol{\lambda}$, and and make the appropriate substitutions, we have:

$$\frac{\mathrm{d}\hat{J}^*}{\mathrm{d}\boldsymbol{m}_t} = \left\langle -\frac{\delta F(\boldsymbol{u},\boldsymbol{m})}{\delta\boldsymbol{m}_t}, \boldsymbol{\lambda}\right\rangle + \frac{\delta J^*}{\delta\boldsymbol{m}_t} \tag{A.18}$$

Equation A.19 provides an alternative method of calculating the gradient – one that does not require calculation of the U x M matrix as in the primal method, but rather the solving of a linear system for the U x 1 adjoint vector $\boldsymbol{\lambda}$, a far less memory-complex task. This adjoint vector is tied to a specific functional, however, and thus if one wants to run a gradient-based descent algorithm on multiple objective functionals, adjoint vectors must be calculated for each objective functional. Yet in the case of PDE-constrained optimization, we generally have a single fixed objective functional and many decision variables; consequently, we use the adjoint approach.