Application of the Double Descent Color Intermittent Diffusion Method to a Variational Data Assimilation Problem

By

Rodrigo Jose Gonzalez Hernandez

Manuela Manetta Ph.D.
Advisor

Department of Mathematics

Manuela Manetta Ph.D.
Advisor

Alessandro Veneziani, Ph.D.
Committee Member

Dan Sinykin, Ph.D.
Committee Member

2022

Application of the Double Descent Color Intermittent Diffusion Method to a Variational Data Assimilation Problem

By

Rodrigo Jose Gonzalez Hernandez

Manuela Manetta Ph.D.
Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences of
Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2022

Abstract

Application of the Double Descent Color Intermittent Diffusion Method to a Variational Data Assimilation Problem

By Rodrigo Jose Gonzalez Hernandez

The Double Descent Color Intermittent Diffusion Method (DD-CID) is a global optimization algorithm that identifies the global minimum of a smooth function through the combination of the Double Descent method and a basin-escape technique with stochastic features. Data Assimilation (DA) is the branch of applied mathematics that involves the optimal blend of mathematical models and observed data. In this work, two data assimilation problems in the form of constrained optimization problems are proposed. Specifically, the goal is to identify some unknown parameters in two boundary-value problems. The DD-CID method is tested against these and numerical results are presented.

Application of the Double Descent Color Intermittent Diffusion Method to a Variational Data Assimilation Problem

By

Rodrigo Jose Gonzalez Hernandez

Manuela Manetta Ph.D.
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2022

Acknowledgments

I would like to thank my thesis advisor, Dr. Manuela Manetta, for all of her support throughout my undergraduate career. I would also like to thank Dr. Alessandro Veneziani and Dr. Dan Sinykin for being on my thesis committee. Finally, thank you to my family and friends for your encouragement.

Contents

List of Figures

List of Tables

Overview

The objective of this work is to test a novel optimization algorithm in a parameter estimation procedure. Numerical optimization problems are ubiquitous in different quantitative fields beyond mathematics such as biology, engineering, and economics. Loosely, its primary goal is to determine the element of the domain of a function that minimizes its value. Moreover, many mathematical models are dependent upon a set of parameters that are unkown in general. However, given some data of the phenomenon that is being modeled, a procedure can be devised so that the parameters are determined. This often leads to the optimization of some function, hence the need of a numerical procedure that performs this task. Most optimization schemes, however, are local in the sense that do not completely explore the function to be optimized. This is challenge since the parameter estimation procedure does not have a unique solution in general. Therefore, a global optimization scheme is highly recommended. This motivates the work in this thesis. More precisely, the Double Descent and Color Intermittent Diffusion (DD-CID) global optimization algorithm is tested against an estimation problem.

The work is organized as follows. Chapter 1 explores Global Optimization. First, a general introduction to optimization, including a discussion of the basic concepts, is given. This section is meant to be Then, the DD-CID is discussed in length. A visual example of the DD-CCID at work is included with the goal of clearly explaining the major steps of the algorithm. Finally, its translation from MATLAB to Python is discussed and numerical results are given.

Chapter 2 explores *Data Assimilation*. This is the broad mathematical framework from which the estimation procedure is derived. Again, a brief introduction to Data Assimilation is given. Then, its particularization to the estimation problem as well as the relevant mathematical tools are discussed. In particular, a discussion of the "misfit functional " as well of its gradient is included. Such functional is precisely the one whose optimization yields the solution of the estimation problem. Then, two specific test problems are discussed. Specifically, a one-dimensional boundary-value problem with unknown parameters and its multi-dimensional counterpart are introduced. Additionally, a lengthy mathematical treatment of these two problems is included. Finally, the specific optimization strategy is detailed.

In Chapter 3, the thesis concludes with the numerical results the as well as the conclusions and discussion of future work.

Chapter 1

Double Descent and Color Intermittent Diffusion for Landscape Exploration

1.1 Global Optimization

Global optimization is one of the oldest and most relevant problems of applied mathematics. This is not a surprise, as it arises in many different fields including the sciences, engineering, and beyond. In many situations, there is a necessity to find the configuration of a system so that some optimal condition is satisfied. It is easy to imagine, for instance, that it is of the utmost importance for business owners to find the amount of labor, machinery, software, etc. that will maximize profit.

A perhaps more sophisticated example can be found in the field of computational biology, specifically in *protein structure prediction.* As described in [APS^{+}22], the objective of this problem is to find the spatial structure of a protein based on the amino acids that composite it. While there are different approaches to this problem, there is evidence that the structure can be successfully predicted by performing a global optimization of some potential energy function, given the amino acid sequence.

It is therefore understandable that massive efforts are put into understanding and developing novel global optimization techniques. In this document, the Double Descent and Intermittent Diffusion (DD-CID) global optimization method is briefly described an used in data assimilation procedure. Such method combines a double-descent technique and a basin-escaping technique in search of global minimizers. However, before further discussing this algorithm, a simple working example is studied in order to provide the basic concepts behind global optimization.

1.1.1 Global Optimization: an introduction through a simple working example

Suppose a function $f(x)$, which will be referred to as the **objective function**, is given. Such function takes a single real number x as input to produce a single real number $f(x)$ as output. For instance, suppose that f is defined as

$$f(x) = \frac{x^4(5x^2 + 6x - 15)}{30} \tag{1.1}$$

Notice that f is a simple *polynomial* function. Its graph around the origin looks as follows.

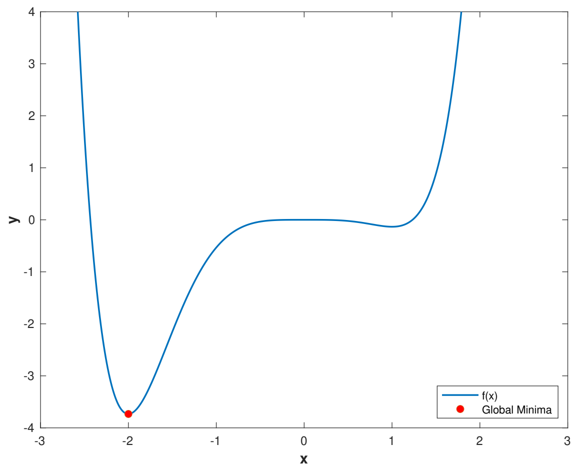

Figure 1.1: The global minima of the objective function.

The goal is to minimize the objective function. More specifically, the *optimization problem* consists of finding a point x_m in the real line such that, for any other point x, the condition

$$f(x_m) \leq f(x) \tag{1.2}$$

is satisfied. Such point will be referred to as the **global minimima** (or **global minimizer**) of f. In other words, the goal is to find the element of the domain of the function that produces the lowest value. By a simple inspection of **??**, it is possible to conjecture that the point $x_m = -2$ satisfies such condition. In most real problems, however, the graph of the objective function is not available, and even if it is, a visual inspection will not be enough to safely conclude a global minima has been found. In other words, a formal mathematical procedure must be developed to solve the optimization problem. In order to introduce the major ideas behind this procedure and many of the relevant definitions that are relevant to this work, imagine that no graph was given for our simple objective function. How could it be formally verified that $x_m = -2$ is in fact the global minima of the cost function?

First, start by noticing the points $x = -2$ and $x = 1$ on the graph. What do these have in common? These points are special since they both are minimal to *some points* in the domain. Specifically, these are the **local minima** (or **local minimizers**) of f. Both points act as global minima of the function, if its domain was to be restricted to a small region around them. In mathematical terms, it is said that a is a local minima of f if there exists a positive real number $\varepsilon > 0$ such that for all x in the interval $(a - \varepsilon, a + \varepsilon)$ the condition

$$f(a) \leq f(x) \tag{1.3}$$

is satisfied. Note that the global minima of the objective function must be itself a local minima. To check this, note that it is always true that $f(x_m) \leq f(x)$, regardless of the choice of $\varepsilon > 0$. This reveals the importance of local minima in the global optimization problem, and provides the foundation of the optimization procedure. Specifically, assuming that at least a reasonably large amount of local minima can be found, choosing the one that achieves the lowest value would be enough to solve the optimization problem. Therefore, the goal is to determine the local minima of a function. This is done by first finding a broader class of points for which the local minima are a subset of: the *critical points* of the objective function.

Intuitively, the **critical points** of f are the set of points for which the graph f is “instantaneously” flat. For the simple objective function that is being considered, the points $x = -2$, $x = 0$, and $x = 1$ satisfy this condition. In order to appreciate this, the following graph contains tangent lines that go through these points.

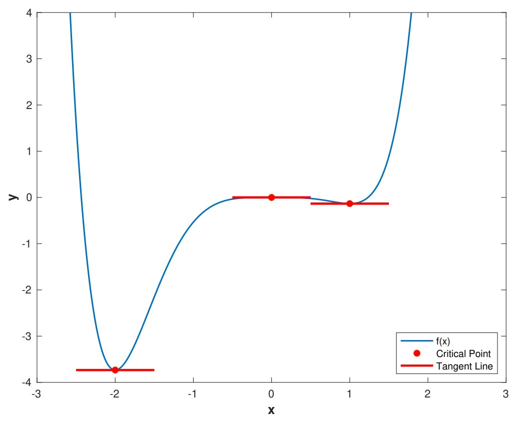

Figure 1.2: The critical points of the objective function.

Mathematically, x_c is a critical point of $f(x)$ if the derivative of f at x_c is zero. If the derivative of f is the function denoted by f', then this condition is equivalent to $f'(x_c) = 0$. For the simple objective function, the derivative is given by

$$f'(x) = x^3(x+2)(x-1) \tag{1.4}$$

It is straightforward to check that $f'(-2) = f'(0) = f(1) = 0$. Note that $x = -2$ and $x = 1$ are the local minima that were previously identified. Therefore, it is true that the the local minima are a subset of the critical points, but the converse is not: not all critical points are local minima. To see this, consider the point $x = 0$. While it is true that $f'(0) = 0$, it is not a local minima.

Generally, critical points may also be **local maxima** (or **local maximizers**) or **saddle points** of a function. The definition of a local maxima is similar to that of a local minima. Specifically, a is a local maxima of a function g if there exists a positive real number $\varepsilon > 0$ such that for all x in the interval $(a-\varepsilon, a+\varepsilon)$ the condition

$$g(x) \leq g(a) \tag{1.5}$$

is satisfied. As an example, consider the following function and its graph

$$g(x) = \frac{x(2x^2 + 3x - 12)}{6} \tag{1.6}$$

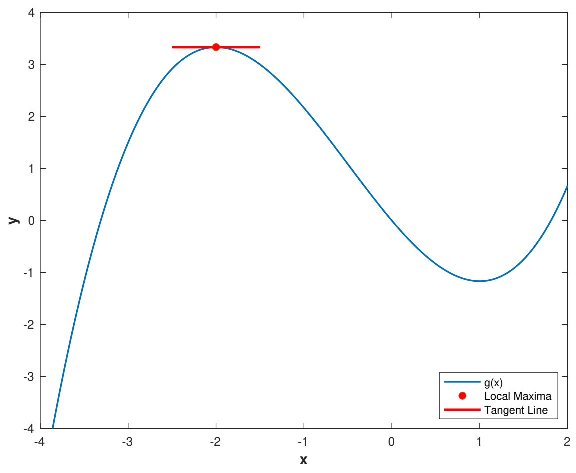

Figure 1.3: The local maxima of the function $g(x)$.

It is possible to appreciate that $a = -2$ is a local maxima. Finally, for one-variable real-valued functions, $f : \mathbb{R} \to \mathbb{R}$, all critical points that are not either local maxima or local minima are called **inflection points** of the function. For instance, consider the function h and its corresponding graph.

$$h(x) = x^3 \tag{1.7}$$

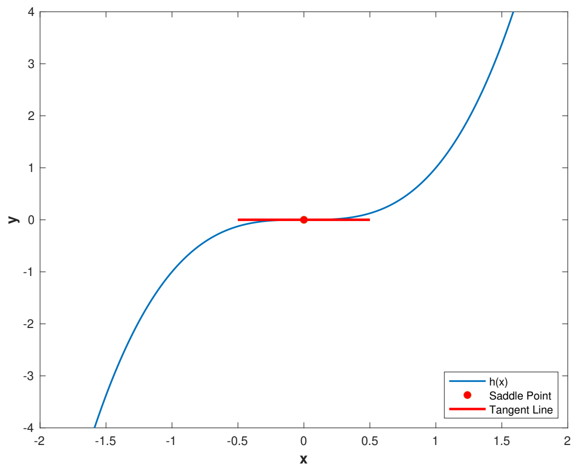

Figure 1.4: The inflection point of the function $h(x)$.

Note that, in this case

$$h(0) > h(x) \quad \text{for all } x < 0 \tag{1.8}$$

$$h(0) < h(x) \quad \text{for all } x > 0 \tag{1.9}$$

Which implies that that no positive real number $\varepsilon > 0$ will make the critical point $x = 0$ fit the definition of local minimum or local maximum of h. Therefore, it is an inflection point. For multi-variable functions, inflections points are generalized as **saddle points**. For instance, consider the function $h : \mathbb{R}^2 \to \mathbb{R}$ defined as

$$h(x_1, x_2) = x_1^2 - x_2^2 \tag{1.10}$$

In higher dimensions, the notion of derivative is generalized with the **gradient of the function**, and is a vector composed with the *partial derivatives* of the function. In mathematical terms, the gradient of h is denoted by ∇h, and its value at the point (x_1, x_2) is given by

$$\nabla h(x_1, x_2) = \begin{bmatrix} h_{x_1}(x_1, x_2) \\ h_{x_2}(x_1, x_2) \end{bmatrix} \quad \text{with} \quad h_{x_i} = \frac{\partial h}{\partial x_i} \tag{1.11}$$

For our h, this is

$$\nabla h(x_1, x_2) = \begin{bmatrix} 2x_1 \\ -2x_2 \end{bmatrix} \tag{1.12}$$

Then, the critical points of h are those points such that the gradient of the function is zero. In other words, the points (x_1, x_2) such that $\nabla h = \vec{0}$. For our function, we have that $(0, 0)$ satisfies this condition. The following figure shows the graph of h around this point

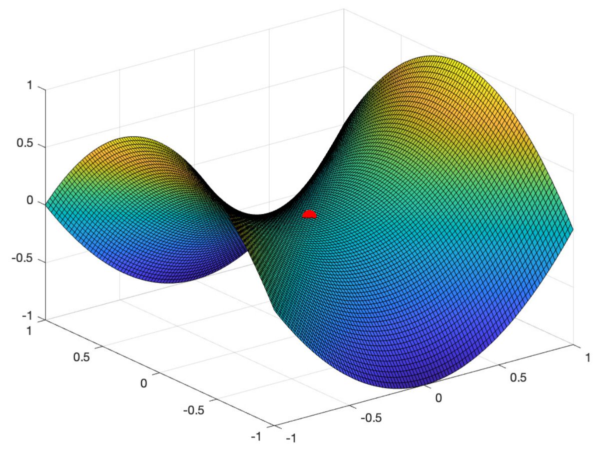

Figure 1.5: The saddle point of the function $h(x)$.

It is possible to see that $(0, 0)$ is not an extremum. Indeed, $(0, 0)$ is a local maximum in the line given by $x_2 = 0$, and a local minimum in the line $x_1 = 0$. Thus, a saddle point.

Now, consider again f, our one-variable objective function. It is now clear that solving the equation $f'(x) = 0$ yields the critical points. It is then necessary to identify which of these are local minima, local maxima, or saddle points. This is done by determining the shape of the function around them. In fact, if the function is **concave up** (shaped like a union symbol " $\cup$ ") around the critical point, then it is a local minima. On the other hand, if the function is **concave down** (shaped like a intersection symbol " $\cap$ ") around the critical point, then it is a local max. Mathematically, this is done by determining the sign of the *second derivative* of the function. This function is denoted by $f''(x)$ and defined as the derivative of the derivative of f. The following theorem describes how it is used to classify critical points.

Second derivative test: Assume that $f''(x)$ is *continuous* at the critical point x_c. Then,

- if $f''(x) > 0$, then x_c is a local minima.
- if $f''(x) < 0$, then x_c is a local maxima.

For the simple objective function, the second derivative is given by

$$f''(x) = x^2(5x^2 + 4x - 6) \tag{1.13}$$

Note that $f''(-2) = 24 > 0$ and $f''(1) = 3 > 0$. Therefore, these two points were correctly identified as local minima. Moreover, note that $f''(0) = 0$. Therefore, this critical point cannot be classified using the second derivative test, as it is inconclusive in this case.

Finally, he general steps for global optimization procedure can be summarized as follows

1. Find the critical points of the objective funciton by solving $f'(x) = 0$.
2. Determine which of these critical points are local minima. For one-variable functions, this can be done following the second derivative test.

3. From the set of local minima, choose the point that produces the lowest smallest value. Set this as best candidate for global minima.

To be clear, this is an oversimplified guide to solving the global optimization problem. Nevertheless, having these steps in mind is helpful as they provide a general scheme that can be followed. Moreover, the process that has been discussed is referred to as a **unconstrained** optimization. That is, x can attain any real value in this problem. However, some constrains can be forced. Imagine, for instance, that it is desired that $x < 0$. While simple, this is a rule that would transform the process into a **constrained** optimization problem. In these cases, the variable is *constrained* to take a specific set of values.

1.1.2 Numerical solution to the simple working example

The previous section described the importance of finding the critical points of a function in the global optimization process. This was achieved by finding the points c such that $f'(c) = 0$. In the previous example, this was done *analytically*. In other words, it was straightforward to determine the critical points from the explicit expression for $f'(x)$ in 1.4. However, in many cases, this can not be done. Instead, a *numerical* procedure to find the critical points is preferred. In general, these are iterative methods that update a point in the domain of the function, and given some iterations of this procedure, it is hoped that the point gets "close enough" to the critical point. That is, a sequence $\{x_1, x_2, x_3 \ldots\}$ is created such that, in the limit, it converges to some critical point c. Mathematically,

$$\lim_{n\to\infty} x_n = c \quad \text{with} \quad f'(c) = 0 \tag{1.14}$$

One of the most popular of such numerical schemes is **Newton's Method**. This is a root-finding algorithm. In other words, if a function $h(x)$ is given, then Newton's

Method produces a sequence $\{x_1, x_2, x_3 \ldots\}$ of points that converges to a a point c such that $h(c) = 0$. The x_{i+1} element of the sequence is determined by the x_i element by following the rule

$$x_{i+1} = x_i - \frac{h(x_i)}{h'(x_i)} \tag{1.15}$$

Note that finding the critical points of $f(x)$ is precisely the same problem as finding the roots of its derivative $f'(x)$. Then, allowing $h(x) = f'(x)$, Newton's Method for critical points is given by

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} \tag{1.16}$$

Note that the rule does not state which point x_0 should be initial element of the sequence. In general, choosing a different starting point will drive the convergence to a different critical point. To illustrate this, consider starting this procedure from the the points $x_0 = -3$, $x_0 = -1$, and $x_0 = 2$ for our simple objective function $f(x)$. The result for the three iterative procedures is shown in the following graph.

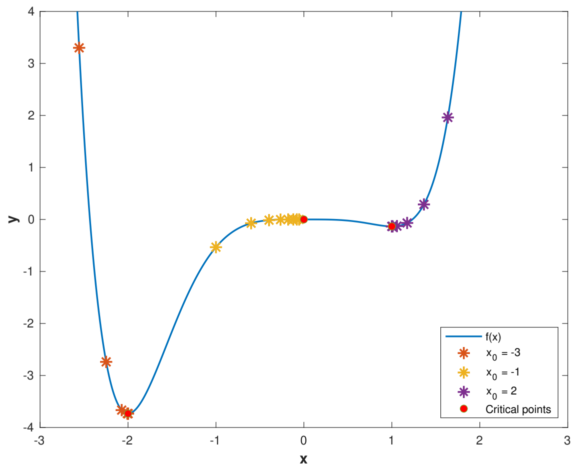

Figure 1.6: Newton's Method to find the critical points of $f(x)$.

The sequence starting from $x_0 = -3$ converged to the critical point at $x_c = -2$, the sequence starting from $x_0 = -1$ converged to the critical point at $x_c = 0$, and the sequence starting from $x_0 = 2$ converged to the critical point at $x_c = 1$. Therefore, repeating Newton's Method for this particular selection of initial points was successful in finding the three critical points of $f(x)$. After this, the second derive test can be used to conclude that $x_c = -2$ and $x_c = 1$ are local minima. Finally, by comparing the values of f at these two points, it is concluded that the global minimum is at $x_m = -2$.

1.1.3 Some difficulties of the global optimization problem

The previous section established two major components of the numerical solution of the global optimization problem. First, the use of a iterative method to find a single critical point, and second, the repetition of this procedure from different starting locations to find all of the critical points. In this section, these two components are revisited in more detail in order obtain a better understanding of some major difficulties in global optimization.

First, the convergence speed Newton's Method and stopping criteria are discussed. To do this, recall that Newton's Method converges to a critical point as the number of iterations goes to infinity. Of course, it is impossible to repeat this process indefinitely, and it must stop after a finite number of steps. This means that the final point of is likely not a critical point, but rather a point that is "close enough" to it. To formalize this notion, let the distance between x_i - the *ith* point produced by Newton's Method - and x_c - the critical point - be measured by $|x_i - x_c|$. Moreover, a positive real $\varepsilon > 0$ is defined to be maximum error allowed. In other words, ε is the **tolerance**, and the iterative procedure finalizes when the condition $|x_i - x_c| < \varepsilon$ is met. Having this notion allows the study of the convergence speed of Newton's Method. This can be illustrated using the example in the previous section. Suppose that the tolerance is set to $\varepsilon = 0.01$. The following tables contain the points produced by Newton's

Method, starting from the initial points $x_0 = -3$ and $x_0 = -1$.

i	x_i	$\lvert x_i + 2\rvert$
0	-3.00	2.00
1	-2.55	0.55
2	-2.25	0.25
3	-2.07	0.07
4	-2.01	0.01
5	-2.00	0.00

Table 1.1: Newton's Method starting from $x_0 = -3$ and converging to $x_c = -2$.

i	x_i	$\lvert x_i - 0\rvert$
0	-1.00	1.00
1	-0.60	0.60
2	-0.39	0.39
$\vdots$	$\vdots$	$\vdots$
12	-0.01	0.01
13	-0.00	0.00

Table 1.2: Newton's Method starting from $x_0 = -1$ and converging to $x_c = 0$.

Notice that, even though the initial error was $|x_0 - x_c| = 1$ in both cases, the sequence starting from $x_0 = -1$ required more than double the number of iterations to reach the allowed tolerance. Moreover, recall the critical point and the error are not known beforehand. This means that a practical stopping criterion must be defined. For instance, consider that a maximum number of iterations is established. While this criterion is simple to implement, it also comes with some risks. Evidently, choosing a small number is desirable as it lowers the computational costs, but on the other hand, it can not too small so that the iterative method does not get close enough to the critical point. For instance, if a small number, say $N = 10$, iterations were allowed in the previous example, Newton's Method would not be *fast enough* to be within a reasonable distance of the $x_c = 0$ if the initial point was $x_0 = -1$. This highlights the importance of using fast iterative schemes: methods that require less iterations to converge to the critical point are referred in general.

Alternatively, an upper bound on the magnitude of the derivative can also be used as stopping criteria. Specifically, a value $\varepsilon > 0$ is chosen and the iterative procedure stops when when the condition $|f'(x_i)| < \varepsilon$ is satisfied. This makes sense intuitively, as the derivative of the cost function is zero at the critical point, and it can be expected to be close to zero for points around it. However, this criterion also has some drawbacks, specially for "flat " regions of the function, as these contain points that are not necessarily close to the critical point, but have small derivative nonetheless. Again, the previous example can be used to illustrate this. For instance, consider the two first elements in the previous two tables. Their error is comparable at $|x_1+2| = 0.55$ and $|x_1-0| = 0.60$, which means that both are at a similar distance of their corresponding critical points. However, the x_1 element of the first sequence has a large derivative of $f'(x_1) = -32.4$, and the x_1 element of the second sequence has a small derivative of $f'(x_1) = 0.5$. Therefore, basing the criterion by the size of the derivative runs the risk of stopping at points that are not close enough to the critical points.

More broadly, this analysis highlights the difficulty of dealing with relatively flat functions. In fact, note that the region around $x_c = 0$ in **??** is flat and Newton's Method considerably slowed down around this point. In general, flat cost functions represent a challenge, and robust numerical algorithms should be able to handle these situations. This is the reason why it is common to test such methods against flat functions. For instance, consider the **Matyas Function** from [JY13]. It is defined $f : \mathbb{R}^2 \to \mathbb{R}$, and given by

$$f(x_1, x_2) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \tag{1.17}$$

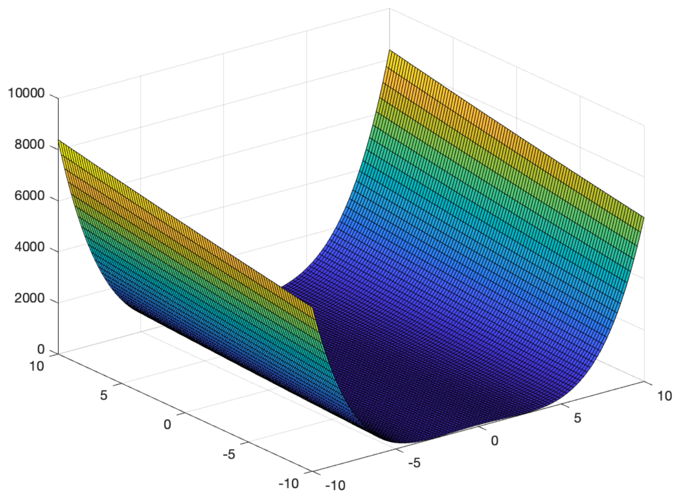

Figure 1.7: The Matyas Function

The only local minima of this function corresponds to the global minima, which is located at $(x_1, x_2) = (0, 0)$. Therefore, in this case, efficiently converging to this critical point is the challenge for the iterative algorithm. Once this point is found, the global optimization problem is solved as there are no other critical points to visit.

Furthermore, recall that in the previous section Newton's Method was repeated starting from different locations so that all critical points were found. In general there is no guarantee that this will always be the case. To illustrate this, suppose that a new set of initial points are selected. In particular, take $x_0 = -1.5$, $x_0 = -0.5$, and $x_0 = 0.5$. Newton's Method will produce the following results

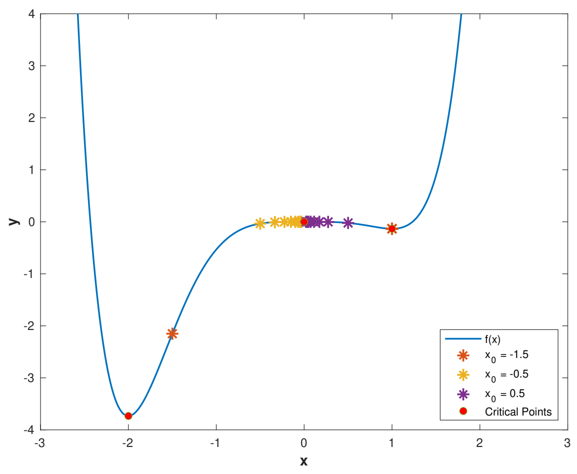

Figure 1.8: Newton's Method on f(x) with a different set of starting points.

In this case, the sequence starting from $x_0 = -1.5$ converged to the critical point at $x_c = 1$, the sequence starting from $x_0 = -0.5$ converged to the critical point at $x_c = 0$, and the sequence starting from $x_0 = 0.5$ also converged to the critical point at $x_c = 0$. Therefore, the critical point at $x_c = -2$ was not visited, which is relevant since this is the global minima of the function. It is also worth pointing out the that the sequence starting at $x_0 = -1.5$ converged, after a single step, to $x_c = 1$. Intuitively, it is reasonable to think that the iterative procedure would converge to the critical point closest to its initial point, which in this case would be $x_c = -2$. This counter example shows that this is not true in general.

More broadly, it also exemplifies another challenging aspect of global optimization: choosing the best initial positions so that the iterative procedure successfully visits different critical points. This is very difficult to know beforehand, specially because cost functions may vary wildly. For instance, consider the **Ackley Function** from [JY13]. This is a two-variable test function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f(x_1, x_2) = -a \exp\left(-b\sqrt{\frac{1}{d}(x_1^2 + x_2^2)}\right) - \exp\left(\frac{1}{d}\left(\cos(cx_1) + \cos(cx_2)\right)\right) + a + 1 \tag{1.18}$$

Where a, b, and c are real parameters. Its graph and its graph is as follows

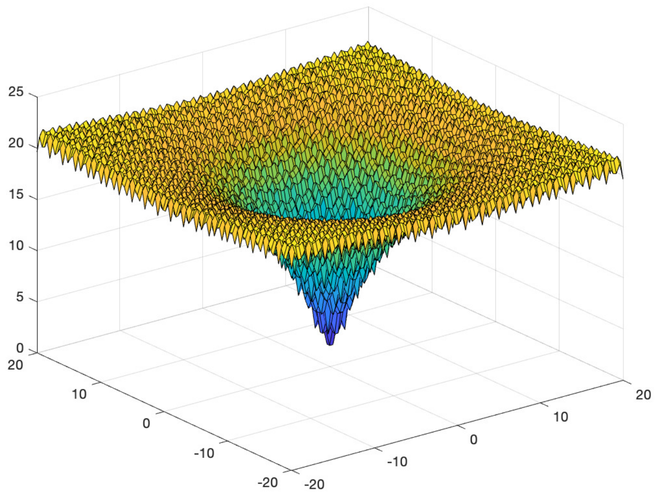

Figure 1.9: The Ackley Function.

Notice that there are many critical points that are not the global minima. Therefore, it is expected to repeat Newton's Method from many different starting locations so that these points are found. Ideally, once a critical point is found, the next starting location should drive Newton's Method to a different critical point, as it would be a waste of computational resources to visit a point that has been already identified.

In conclusion, there are two main challenges that a numerical algorithm must address. First, it should implement a robust iterative scheme to converge to a critical point. Second, after finding the critical point, it should restart the iterative scheme so that a critical point a different critical point is found. In the following section, an algorithm with this characteristic is discussed.

1.2 DD-CID

The **Double Descent and Color Intermittent Diffusion (DD-CID)** is the global optimization algorithm considered in this thesis. It was originally introduced in [DMZ20]. It consists of two major processes that are repeatedly executed one after the other: local search and landscape exploration of the objective function. In the former, either Damped Newton or Double Descent is used to reach a critical point. In the latter, a basin-escaping approach with stochastic features is used to move away from the critical point so that the function is explored. These two major processes are discussed in the following sections.

1.3 Local search

During the local search process, the DD-CID algorithm identifies a critical point of the objective function. There are two main cases: the search of a saddle point and the search of a local minimum. Both of these are accomplished through iterative algorithms. These may be written in the following way

$$x_{i+1} = x_i - h_i v_i \tag{1.19}$$

where v_i is the *search direction* and $h_i > 0$ is a parameter that modifies the step size. Converging to a specific type of critical point - saddle points or local minimum - may be achieved by modifying the choice of v_i. This is considered in the following sections.

1.3.1 Local search: saddle point

Newton's method is a popular iterative scheme used to find critical points. For multi-variable objective functions $f : \mathbb{R}^n \to \mathbb{R}$, it is given by

$$x_{i+1} = x_i - H^{-1}(x_i)\nabla f(x_i) \tag{1.20}$$

Where $\nabla f(x)$ is the gradient and $H(x)$ is Hessian of the objective function. These are the generalizations of the first and the second derivative. We can see that Newton's direction is $v_i = H^{-1}(x_i)\nabla f(x_i)$. This is a "pure form" of Newton's method, and it is *not* the one implemented by DD-CID. Instead the algorithm uses **Damped Newton's method** to converge to a saddle point. This variation of Newton's method is implementing by choosing some $h_i > 0$ and allowing

$$x_{i+1} = x_i - h_i(H^{-1}(x_i)\nabla f(x_i)) \tag{1.21}$$

Again, h_i modifies the step size. Its specific choice will be discussed in the following section. Moreover, it is worth noting that Newton's method converges to critical points rather than to saddle points specifically. Therefore, it is possible that the method may identify a local minimum. This is not a problem since the discovery of local minima is precisely one of the goals of the optimization algorithm.

1.3.2 Local search: local minimum

Now we consider the local search of a minimum. The DD-CID algorithm uses **Double Descent method** to achieve this. To introduce it, we again recall that there is no guarantee that Newton's method will converge to a local minimum. Therefore, it would be ideal to use an iterative method that takes a step in a **descent direction**.

A descent direction is a direction v such that $f(x + hv) < f(x)$ for some $h > 0$. Thus, moving in such direction would force a method to "go downhill" - that is, decrease - the objective function and converge to a minimum instead of other critical points. Moreover, a descent direction is not unique, and if v is a descent direction, then $(\nabla f(x))^T v < 0$. It is common to take the gradient-descent choice, that is

$v = -\nabla f(x)$. This leads to the **Steepest Descent** method

$$x_{i+1} = x_i - h_i \nabla f(x_i) \tag{1.22}$$

Where we choose the parameter $h_i > 0$ sufficiently small such that $f(x_{i+1}) < f(x_i)$, assuming that $\nabla f(x_i) \neq \vec{0}$. Therefore, this method achieves the desired "descent" over the function. However, it is also very slow: compared to other methods, it may require many iterations to converge to a local minimum.

The **Double Descent method** combines these two previous methods. In order to introduce it, first we define the "auxiliary potential" $F : \mathbb{R}^n \to \mathbb{R}$ as $F(x) = \frac{1}{2}(\nabla f(x))^T(\nabla f(x))$. Note that $\nabla F(x) = H(x)\nabla f(x)$, which implies that Newton's direction is a descent direction of this auxiliary potential $F(x)$.

The Double Descent direction is a descent direction for both f and F, hence its name. This combination ensures that we converge quickly to a local minimum. If the Hessian is positive definite then the Double Descent direction corresponds to Newton's direction. If the Hessian is indefinite, then it is replaced with H_+, the closest positive semidefinite matrix to it. More precisely, if $H = V\Lambda V^T$ is the diagonalization of H, and $\Lambda = \text{diag}(\Lambda_+, \Lambda_0, \Lambda_-)$ is the diagonal matrix containing its eigenvalues, then $H_+ = V\Lambda^+V^T$ with $\Lambda^+ = \text{diag}(\Lambda_+, 0, 0)$. In other words, H_+ is the projection of $V\Lambda V^T$ onto the positive eigenvalue subspace. Then, in these two cases the Double Descent method is given by

$$x_{i+1} = x_i - h_i(H_+(x_i))^\dagger \nabla f(x_i) \tag{1.23}$$

Where $M^\dagger$ is the *pseudoinverse* of the matrix M. Note that, if H is positive definite then $H = H_+$ and the previous is precisely Newton's Method.

1.4 Landscape exploration

Once a critical point has been found, the local search must be restarted. Ideally, this will be done from a point that drives the iterative procedure to converge to a different critical point. This is achieved by the **landscape exploration of the function**. As in the local search, there are two main processes, namely, the local minimum to saddle point step, and the the saddle point to local minimum step.

1.4.1 Local minimum to saddle point

The **local minimum to saddle point** step is achieved by first escaping the **basin of attraction** of the local minimum. This is a relatively steep region around it. A local search algorithm that starts on the basin of attraction a minimum will converge to it. After escaping it, the local search process described in the previous section is performed: Damped Newton's method is used to converge to a saddle point. Specifically, there are three basic steps to go from a minimum to a saddle.

First, we activate the *color noise diffusion.* This is the process by which the method escapes the basin of attraction of the minimum. We take some $\alpha > 0$, a parameter chosen by the user of the algorithm, and perform an initial "kick" out of the minimum by allowing

$$x_1 = x_0 + \alpha\sigma(x_0)W \tag{1.24}$$

Here, x_0 is the local minimum, $W = N(0,1)^T$, and $\sigma(x_i) = -v_1(x_i)v_1(x_i)^n$ is a rank 1 matrix, where $v_1(x_i)$ is the eigenvector corresponding to the largest eigenvalue of the Hessian at x_i. The term $\sigma(x_i)W$ corresponds to a "diffusive step", and the choice of α will amplify its size. After this, we proceed by allowing

$$\hat{x}_{i+1} = x_i - h_i H^{\dagger}(x_i)\nabla f(x_i) \tag{1.25}$$

to be the step that would be taken with Newton's method. The choice of h_i is made such that $|F(\hat{x}_{i+1})| < |F(x_i)|$, which ensures that we are approaching a critical point. This Newton's step is then modified by adding the diffusive term. Namely,

$$x_{i+1} = \hat{x}_{i+1} + \alpha\sqrt{h_i}\sigma(x_i)W \tag{1.26}$$

The diffusion process stops once the Hessian contains some negative eigenvalues. In other words, the iterations continue as long as x_i is in a region where the Hessian is positive definite and stops when it enters one where it is indefinite. Finally, Damped Newton's method is used to converge to a saddle point. This is achieved by allowing $x_{i+1} = \hat{x}_{i+1}$. In other words, the diffusive term is no longer considered.

1.4.2 Saddle point to local minimum

The **saddle point to local minimum** step is performed in an analogous way. The process starts by first escaping the basin of attraction of a saddle. After this, the local search process described in the previous section is performed: Double Descent is used to converge to another local minimum. More specifically, there are three basic steps to go from a saddle to a minimum.

First, take some $\alpha > 0$ and again perform an initial "kick" out from the saddle by

$$x_1 = x_0 + \alpha\sigma(x_0)W \tag{1.27}$$

Here, x_0 is the saddle point and $\sigma(x_i) = -v_n(x_i)v_1(x_i)^T$ is a rank 1 matrix, where $v_n(x_i)$ is the eigenvector corresponding to the smallest (in this case, the most negative) eigenvalue of the Hessian at x_i. Similar to the previous case, the term $\sigma(x_i)W$ corresponds to a "diffusive step", and the choice of α amplifies its size.

Then, **Diffused Double Descent** is performed to escape the basin of attraction of the saddle, while also descending over the objective function. To do this, allow

$$\hat{x}_{i+1} = x_i - h_i H_+^\dagger(x_i)\nabla f(x_i) \tag{1.28}$$

to be the step that would be taken with double descent. Here, the choice of h_i depends on $\nabla f(x_i)$ has a meaningful component on the direction of $V_+(x_i)$ - the matrix whose columns are the eigenvectors of $H(x_i)$ which correspond to positive eigenvalues. If there is a meaningful component, then h_i is selected so that both $|f(\hat{x}_{i+1})| < |f(x_i)|$ and $|F(\hat{x}_{i+1})| < |F(x_i)|$. If there is not, then h_i is selected so that $|f(\hat{x}_{i+1})| < |f(x_i)|$. Therefore, this last inequality holds in both cases, and it is guaranteed that objective function always decreases in absolute value. Then, Diffused Double Descent is achieved by adding a diffusive term as follows

$$x_{i+1} = \hat{x}_{i+1} + \alpha\sqrt{h_i}\sigma(x_k)W \tag{1.29}$$

The diffusive process continues as long as x_i is in a region where the Hessian is indefinite and stops when it enters one where it is positive definite. Finally, Double Descent is used to locate a local minimum nearby. Similarly to the previous step, this is achieved by allowing $x_{i+1} = \hat{x}_{i+1}$ so that the diffusive term is no longer used.

1.5 DD-CID at a glance

The DD-CID algorithm starts with an initialization step, and then enters a main loop. During the **initialization**, and initial point is randomly chosen within a region of interest. Such point may be chosen from *some a priori* information of the objective function. Then, Double Descent is used to find a local minimum. This initial minimum is stored in a table that will be used to contain the critical points that have been visited.

Then the **main loop** starts. First, a random point is chosen from the table. Note that in the first iteration it only contains the initial minimum. Then, the process described in the previous section is followed. That is, a diffusive process is used to escape the basin of attraction of the critical point, and then, a local search is used to find the next critical point. This new point is stored in the table. The main loop is repeated for a number of times. Finally, the local minimum with smallest value is extracted from the table of critical points. This is the candidate for global minimum proposed by the algorithm.

1.6 DD-CID visualized

In this section we showcase the DD-CID as it explores the simple test function 3.1 discussed in the previous section.

1.6.1 Contour plot and Hessian regions

The following is the contour plot of the test function. There are three critical points: $(-1, 0)$, $(1, 0)$, and $(0, 1)$. The first two are global minima, and the last is a saddle point.

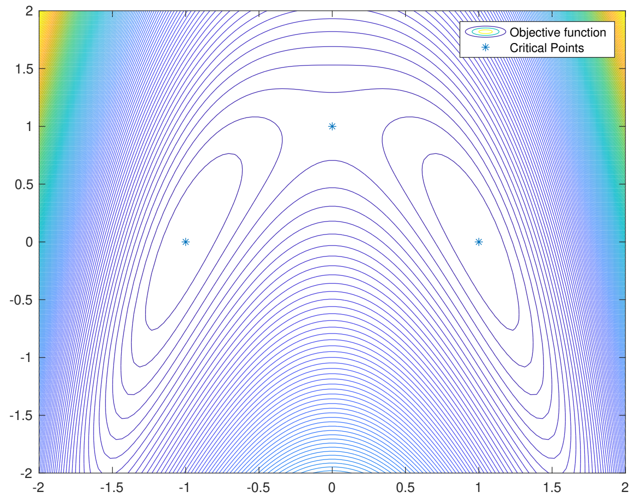

Figure 1.10: Simple Test function

The following shows the regions the “Hessian regions” of the function. The Hessian of the objective function is positive-definite in the blue region and indefinite in the white.

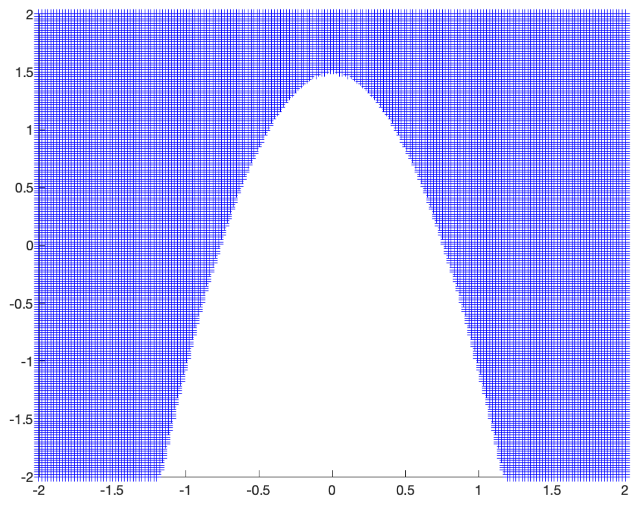

Figure 1.11: Hessian regions

1.6.2 Finding an initial minimum through Double Descent

The first step is to find an initial minimum point. The starting location was chosen to be $x_0 = (-0.5, 0)$. The steps of taken by the Double Descent algorithm to converge to the minimum at $x_{min} = (-1, 0)$ are shown in the following figure.

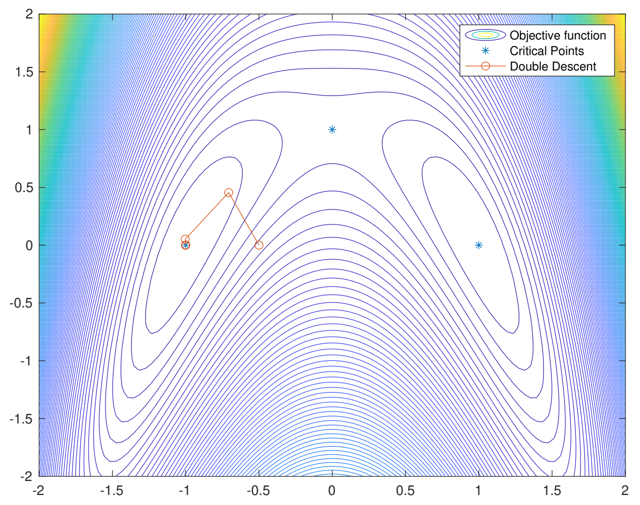

Figure 1.12: Initial Double Descent.

1.6.3 Escaping the basin of attraction of the minimum through color noise diffusion

Now that an initial minimum is known, the DD-CID proceeds to escape its basin of attraction. As described in the previous section, this is achieved by a diffusive process that stops when a Hessian-indefinite region is reached. Indeed, the following figure shows that the final point is near $x = (0, -0.5)$, which is in a indefinite region as shown in Figure **??**.

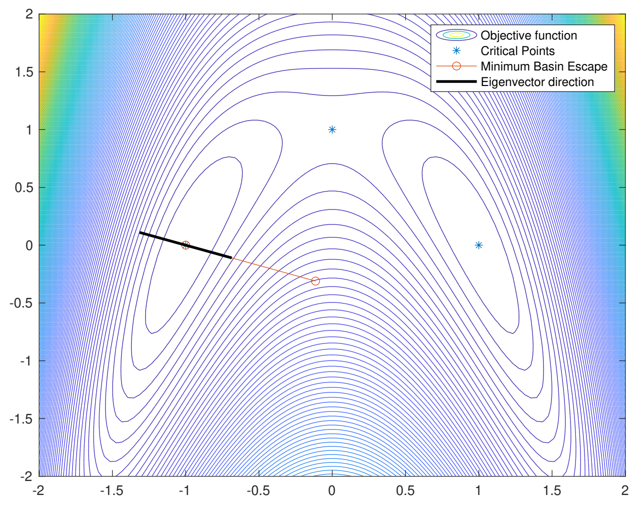

Figure 1.13: Escaping the basin of attraction of the minimum.

Note that, while the process has a random component, the search direction is generally along the span of the eigenvector corresponding to the largest eigenvalue.

1.6.4 Finding a saddle point through Damped Newton

Now the goal is to find a saddle point nearby. This is achieved through Damped Newton. The following figure shows how this scheme starts at the final location of the previous step and converges to the saddle point at $x_{sad} = (0, 1)$.

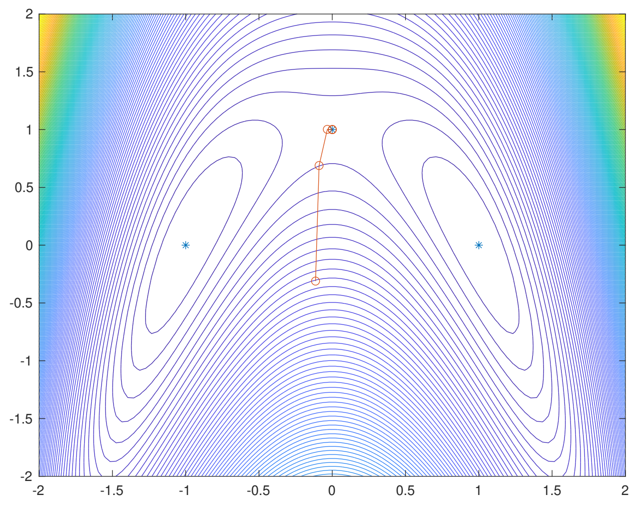

Figure 1.14: Damped Newton to saddle point.

1.6.5 Escaping the basin of attraction of the saddle through Diffused Double Descent

At this point the DD-CID algorithm has found two critical points. To keep exploring the function, it randomly selects one of these points, and starts escaping its basin of attraction. For simplicity, we may assume that the saddle point was chosen. The basin-escape is then performed through Diffused Double Descent. This process stops when a positive-definite region is reached. We see in the following figure that the final point is near $x = (1.5, 1)$, which is in a positive-definite region as shown in Figure **??**.

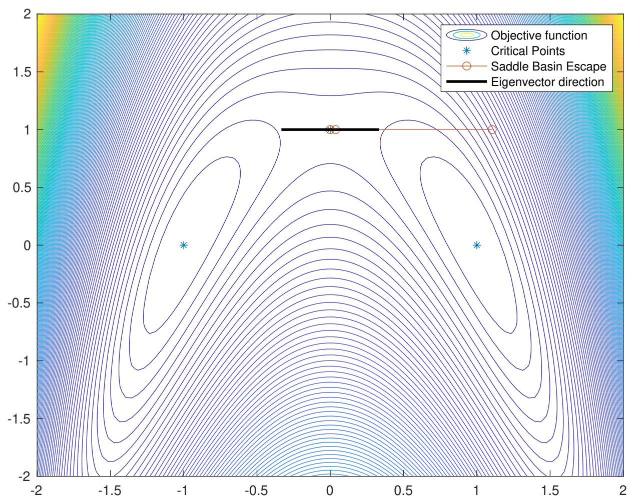

Figure 1.15: Escaping the basin of attraction of the saddle.

Note that, while the process has a random component, the search direction is generally along the span of the eigenvector corresponding to the smallest eigenvalue.

1.6.6 Finding a local minimum point through Double Descent

Finally, the DD-CID performs a final local search in which the second minimum is identified. This is achieved through Double Descent, and the steps taken are displayed in the following figure.

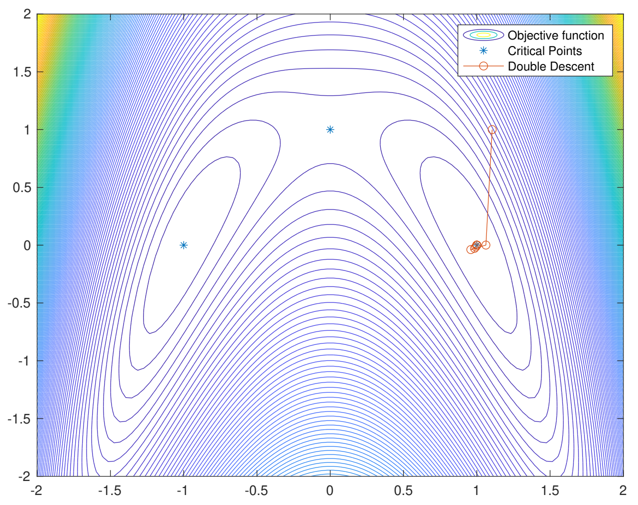

Figure 1.16: Double Descent to second minimum.

All critical points have been found, the minimum that produces the lowest value of the objective function is chosen as global minimum. In this case, these would be the two minima found.

Chapter 2

Data Assimilation and Global Constrained Optimization

2.1 Data Assimilation: the big picture

Data assimilation is the branch of applied math that concerns with the combination of mathematical models and data or measurements to improve the knowledge of real problems.

The early development of DA was mainly motivated by meteorology, specifically, in Numerical Weather Prediction (NWP). As stated in [Nav09], atmospheric DA was born from the need of estimating the initial conditions of the initial-value problems posed by NWP. Moreover, in [JY13], DA is described as the mathematical assimilation of available observations with forecast models in order to provide accurate weather analyses. The need of this came from the availability of more data to combine with mathematical models in a sort of continuous improvement. The sequential improvement of knowledge provided by models with available data is called "filtering". Mathematical models are bound to be imperfect descriptions of natural phenomena. They are often limited by simplifying assumptions and incomplete knowledge in the form of inaccurate or unknown initial conditions, boundary conditions, parameters, etc. On the other hand, observed data is not fully reliable, as it is corrupted by some noise or random perturbations. Therefore, DA concerns the optimal blend of these two components to obtain more reliable quantitative analyses.

There are several techniques in DA, most of them belong to two groups: *sequential methods* (e.g., Kalman Filtering) and *variational techniques*. In the first case, the assimilation happens at each time step of an evolutionary problems, the second group relies on methods of constrained optimization. The latter comes to play in the frame of using the parameter to estimate as a control variable to minimize the mismatch between the observed data and the numerical predictions.

In recent years, DA has been applied to the medical sciences. For instance, [BDPV14] offers an introduction to DA applied to cardiovascular and blood-flow problems. In [YV15], DA is applied to the field of electrophysiology. Specifically, a variational DA approach is used to estimate the electrical conductivity of the cardiac tissue from available potential measures. The idea is to extract from the (surface) measures a knowledge about the status of the (volumetric) cardiac tissue. This knowledge may eventually detect pathological regions (called "scars") or help identifying the optimal implementation of therapies like cardiac re-synchronization therapy (pacemaker) or ablation (burning some small portion of tissue to restore a physiological potential propagation).

In [YV15], the authors pursue a variational approach, where *optimization* comes to play [Gun02]. We basically consider a control problem, by introducing (i) an objective functional to minimize; (ii) variables that define the state of our system (the cardiac tissue), like the intra-cellular, extra-cellular potentials and their difference (called transmembrane potential); (iii) variables to implement the control, i.e. the minimization, in our case the parameters to estimate (the cardiac conductivity); (iv) a problem stating the connection between the control and the state variables. In electrophysiology, the point (iv) is represented by the so called “monodomain equations”.

With these ingredients, we can formalize a generic control (DA) problem: “find the control variables that minimize the optimization functional, under the constraint of the equation(s) in (iv).”

We summarize the ingredients of a control problem:

- State variables
- Control variables
- An objective functional
- Constraints the state and control variable must satisfy.

In continuity with the previous Chapter, we do have an optimization problem here (i.e. something to minimize), yet we have a constraint too.

A classical approach to consider the constraint is to pursue the optimization of a modified functional, incorporating the constraint through a coefficient called *Lagrange multiplier*. In our case, the constraint is not just an algebraic equation, it is an initial-boundary value problem (monodomain).

How to incorporate an IBVP into the Lagrange multiplier constrained optimization will be addressed in detail in the next section.

2.2 Parameter estimation

In this section we introduce the major concepts behind the parameter estimation procedure which is relevant to this work. Suppose we have a **state variable** u which satisfies a differential equation with some boundary conditions. For instance, suppose that u satisfies

$$\alpha u''(x) + u(x) = 0 \qquad x \in \Omega \tag{2.1}$$

$$u(x) = 0 \qquad x \in \partial\Omega \tag{2.2}$$

Here α plays the role of a parameter that we need to estimate. The following procedure is inspired on the one described in [BDPV14] and [Gun02]. We represent the boundary value problem with the general abstract notation

$$F(\alpha, u) = 0$$

This is called the **state equation**. We suppose to have observations of u, that we denote by d. In general, d is available only in some points x_i. Here we assume to have observations for all $x \in \Omega$, for sake of simplicity. Generally, we can postulate that the measures are affected by some noise.

One possible way of estimating the parameter α to explain the observations δ is to resort to a *variational* approach. To this aim, we introduce a **mismatch functional** defined as

$$\mathcal{J}(\alpha, \mu) = \frac{1}{2}\int_0^1 (u(\alpha) - d)^2$$

This functional measures how “far away” a generic solution u is from the reference d. It is important to emphasize that u depends on α, as taking different values of this parameter will produce different solutions as the state equation is solved.

In general, the dependence of the state variable u on the parameter is not trivial. However, we can consider the state equation as a constraint in the optimization process, and recast the parameter identification into the form of a constrained optimization. Should an explicit formula stating the dependence $u = u(\alpha)$ be available, clearly we can proceed with a regular free minimization procedure (like the one considered in the previous Chapter). In this case, we notice that the functional J is non-negative (closer to 0 is our functional and better the optimal solution is). However, in general the dependence of $\Gamma on \alpha$ may be such that J is not convex. This may impair the theoretical results for a specific minimization. To solve the constrained minimization, we resort to a Lagrange multiplier approach. However, the constraint is described by a Boundary Value Problem, and this makes the formalization of the Lagrange multiplier method quite complicated. As a matter of fact, we define the Lagrange functional as

$$\mathcal{L}(\alpha, u, \lambda) = \mathcal{J}(\alpha, u) - \langle \lambda, F(\alpha, u) \rangle \tag{2.3}$$

Where λ is the **adjoint variable** (i.e. the lagrange multiplier) and the notation $\langle , \rangle$ represents formally an inner product. In this way, the solutions to the constrained optimization problem are the critical points of $\mathcal{L}$. In other words, the **Gâteaux derivatives** of $\mathcal{L}$ with respect of each of the variables is set to zero, and the resulting system is be solved.

2.2.1 The state equation

Taking the derivative with respect of the adjoint variable leads to the state equation.

$$\frac{D\mathcal{L}}{D\lambda} = 0 \implies F(\alpha, u) = 0$$

Intuitively, this makes sense as we require that the solution of the optimization problem to be a solution to the state equation.

2.2.2 The adjoint problem

Taking the derivative with respect of the state variable leads to the adjoint problem.

$$\frac{D\mathcal{L}}{Du} = 0 \implies \frac{D\mathcal{J}}{Du} = \left\langle \lambda, \frac{D\mathcal{F}}{Du} \right\rangle$$

Solving this equations leads to λ, the adjoint variable. This can then be used to compute $D\mathcal{F}/D\alpha$ using the expression found through the following condition.

2.2.3 The optimality condition

Taking the derivative with respect of the control variable leads to the optimality condition.

$$\frac{D\mathcal{L}}{D\alpha} = 0 \implies \frac{D\mathcal{J}}{D\alpha} = \left\langle \lambda, \frac{D\mathcal{F}}{D\alpha} \right\rangle$$

This is equation is useful as it gives a way to compute the derivative of the mismatch functional with respect of the control variable. Its specific use will be seen in the following sections.

Solving this system would, in theory, give us the solution to the constrained optimization problem, which would ultimately allow us to find the unknown control variable α^*. However, this is rarely done. Instead, the derivative of $\mathcal{J}$ with respect of the control variable(s) is used to numerically optimize this function. Nonetheless, the ideas presented here provide the mathematical foundation of the next sections.

2.2.4 Two test problems

The present work explores two simple parameter estimation problems. In both of them, a differential equation is given along with a solution corresponding to some prechosen values of the parameters. Reconstructing those values will provide evidence of the soundness of the mathematical argument as well as the strength of the numerical procedure. In particular, it will showcase the potential of the DD-CID as a general-purpose optimization algorithm. This will be shown in the next chapter. Before that, however, it is necessary to explore the mathematics behind the estimation. Specifically, the next two sections introduce the two test problems, and carefully detail the derivation of the gradient of $\mathcal{J}$, which is a necessary input for the DD-CID algorithm.

2.3 One-dimensional problem

2.3.1 Boundary value problem

For the one-dimensional problem, we consider the following differential equation.

$$-\mu u''(x) + \beta u'(x) = -\beta \qquad x \in \Omega \tag{2.4}$$

$$u(x) = 0 \qquad x \in \partial\Omega \tag{2.5}$$

Where $\mu > 0 \in \mathbb{R}$, $\beta \in \mathbb{R}$, and $\Omega = [0, 1]$. The task is to estimate μ and β. The general solution is

$$u(x) = \frac{1 - e^{\beta x/\mu}}{1 - e^{\beta/\mu}} - x$$

Note that it depends on the ratio β/μ of the parameters rather than their specific values. If we let $\mu = \beta$ by, for instance, allowing $\mu^* = 1$ and $\beta^* = 1$, then we have

$$d(x) = \frac{1 - e^{x}}{1 - e} - x$$

This is the **reference function** that we would like to approximate.

2.3.2 Function to be optimized

Treating d as known, we try to estimate μ and β that correspond to this function, i.e. $(\mu^*, \beta^*) = (1, 1)$. For this we consider the following functional that measures the "mismatch" between d, the reference, and u, the solution to Equation 2.12.

$$\mathcal{J}(\mu, \beta) = \frac{1}{2} \int_0^1 (u - d)^2$$

Note that u depends on the values of (μ, β) that are used to solve the differential equation. By minimizing $\mathcal{J}$, the mismatch between d and u goes to zero, and the parameters that correspond to d are found. In other words,

$$(\mu^*, \beta^*) = \arg \min_{(\mu,\beta)} \mathcal{J}(\mu, \beta)$$

Note that the solution is *not* unique. In fact, $d(x)$ is always obtained as the state problem is solved with a pair of parameters such that $\mu = \beta$, therefore giving the functional an infinite number of global minima, and the optimization problem an infinite number of solutions.

2.3.3 Derivation of the gradient

The gradient Let the gradient of $\mathcal{J}$ be denoted by

$$\nabla \mathcal{J} = \left(\frac{D\mathcal{J}}{D\mu}, \frac{D\mathcal{J}}{D\beta} \right)$$

Adjoint operator Consider the adjoint operator. Call it

$$\left. \frac{\partial F}{\partial u} \right|_u^* (p)$$

By definition

$$\left\langle v, \frac{\partial F}{\partial u}\bigg|_u^* (p) \right\rangle = \left\langle \frac{\partial F}{\partial u}\bigg|_u (v), p \right\rangle$$

for any v and p. Here,

$$F(u) = -\mu u'' + \beta u' - \beta = 0$$

is the functional related to the state problem, and

$$\frac{\partial F}{\partial u}\bigg|_u (v) = -\mu v'' + \beta v'$$

is the Fréchet operator of F applied to v. Then, we have

$$\int_0^1 p \frac{\partial F}{\partial u}\bigg|_u^* (v) = \int_0^1 (-\mu v'' + \beta v')p$$

It is possible to show through integration by parts that

$$\int_0^1 (-\mu v'' + \beta v')p = \int_0^1 v(-\mu p'' - \beta p')$$

Therefore, the adjoint operator is

$$\frac{\partial F}{\partial u}\bigg|_u^* (p) = -\mu p'' - \beta p'$$

Adjoint problem The adjoint problem requires to find p such that

$$\left\langle \frac{\partial F}{\partial u}\bigg|_u^* (p), v \right\rangle = \frac{\partial \mathcal{J}}{\partial u}\bigg|_u (v)$$

for all v. Given the definition of $\mathcal{J}$, we have

$$\frac{\partial \mathcal{J}}{\partial u}\bigg|_u (v) = \int_0^1 (u - d)v$$

which gives the weak formulation of the adjoint problem

$$\int_0^1 (-\mu p'' - \beta p')v = \int_0^1 (u-d)v$$

since this must hold true for any v, this is equivalent to the strong form

$$-\mu p'' - \beta p' = u - d \qquad x \in \Omega \tag{2.6}$$

$$p(x) = 0 \qquad x \in \partial\Omega \tag{2.7}$$

which is the problem to be solved in order to find p.

Partial derivatives We use the chain rule to compute the partial derivatives. For instance, working with β we have

$$\frac{D\mathcal{J}}{D\beta} = \frac{\partial \mathcal{J}}{\partial u}\left(\frac{\partial u}{\partial \beta}\right) + \frac{\partial \mathcal{J}}{\partial \beta}$$

Note that

$$\frac{\partial \mathcal{J}}{\partial \beta} = 0$$

since there is no explicit dependency of $\mathcal{J}$ on β. This will not be true if some terms are added for a regularazing effect. Going back, considering first the adjoint problem, and then the adjoint definition

$$\frac{D\mathcal{J}}{D\beta} = \frac{\partial \mathcal{J}}{\partial u}\left(\frac{\partial u}{\partial \beta}\right) = \left\langle \left.\frac{\partial F}{\partial u}\right|_u^* (p), \frac{\partial u}{\partial \beta} \right\rangle = \left\langle p, \left.\frac{\partial F}{\partial u}\right|_u \left(\frac{\partial u}{\partial \beta}\right) \right\rangle \tag{2.8}$$

Note that since

$$F(u) = 0 \tag{2.9}$$

by chain rule, it is true that

$$\frac{\partial F}{\partial u}\left(\frac{\partial u}{\partial \beta}\right)+\frac{\partial F}{\partial \beta}=0 \tag{2.10}$$

Therefore, the partial derivative is given by

$$\frac{D\mathcal{J}}{D\beta}=\left\langle\ p, \left.\frac{\partial F}{\partial u}\right|_u \left(\frac{\partial u}{\partial \beta}\right)\right\rangle=\left\langle\ p, -\frac{\partial F}{\partial \beta}\right\rangle \tag{2.11}$$

Finally, considering F, this last term is

$$\frac{\partial F}{\partial \beta}=u'-1$$

A similar procedure can be followed for μ.

2.3.4 Summary

To optimize $\mathcal{J}$, we need its gradient $\nabla\mathcal{J}$ evaluated at any value of the parameters (μ, β). To calculate it, we follow these steps

1. Given values μ and β, find u by solving the state problem

$$\begin{aligned} -\mu u''(x)+\beta u'(x) &= -\beta \qquad x\in\Omega \\ u(x) &= 0 \qquad x\in\partial\Omega \end{aligned}$$

   This solved numerically - using a finite difference approach.

2. Using the same solver, find p by solving the adjoint problem

$$-\mu p''(x) - \beta p'(x) = u(x) - d(x) \qquad x \in \Omega$$
$$p(x) = 0 \qquad x \in \partial\Omega$$

3. Compute the gradient of $\mathcal{J}$ with the formulas

$$\frac{D\mathcal{J}}{D\mu} = \int_0^1 pu'' \tag{2.12}$$
$$\frac{D\mathcal{J}}{D\beta} = -\int_0^1 p(u'-1) \tag{2.13}$$

The integration is done using the trapezoid rule.

2.4 Multi-dimensional problem

2.4.1 Boundary value problem

For the multi-dimensional problem, we consider the following differential equation.

$$-\mu\Delta u + \vec{\beta}\cdot\nabla u = f \qquad x \in \Omega \tag{2.14}$$
$$u(x) = 0 \qquad x \in \delta\Omega \tag{2.15}$$

Where

$$f = y(y-1)(30x-17) + x(x-1)(40y-22)$$

and $\Omega = [0,1]\times[0,1]$, $\mu > 0 \in \mathbb{R}$, and $\vec{\beta} = (\beta_x, \beta_y) \in \mathbb{R}^2$. The task is to estimate μ and $\vec{\beta}$. If we let $\mu^* = 25$, $\vec{\beta}^* = (15, 20)$, the exact solution reads

$$d(x,y) = x(x-1)y(y-1) \tag{2.16}$$

This is the reference function that we would like to approximate.

2.4.2 Function to be optimized

Treating d as known, we try to estimate μ and $\vec{\beta}$ that correspond to this function, i.e. $(\mu^*, \beta_x^*, \beta_y^*) = (25, 15, 20)$. For this we consider the following functional that measures the "mismatch" between d, the reference, and u, the solution to Equation 2.14.

$$\mathcal{J}(\mu, \beta_x, \beta_y) = \frac{1}{2}\int_\Omega (u-d)^2 \tag{2.17}$$

Note that u depends on the values of (μ, β_x, β_y) that are used to solve the differential equation. By minimizing $\mathcal{J}$, the mismatch between d and u goes to zero, and the parameters that correspond to d are found. In other words,

$$(\mu^*, \beta_x^*, \beta_y^*) = (25, 15, 20) = \arg\min_{(\mu,\vec{\beta})} \mathcal{J} \tag{2.18}$$

2.4.3 Derivation of the gradient

The gradient Let the gradient of $\mathcal{J}$ be denoted by

$$\nabla\mathcal{J} = \left(\frac{D\mathcal{J}}{D\mu}, \frac{D\mathcal{J}}{D\beta_x}, \frac{D\mathcal{J}}{D\beta_y}\right) \tag{2.19}$$

Adjoint operator definition Similarly, consider the adjoint operator. Let it be denoted by it

$$\left.\frac{\partial F}{\partial u}\right|_u^* (p)$$

By definition

$$\left\langle v, \left.\frac{\partial F}{\partial u}\right|_u^* (p) \right\rangle = \left\langle \left.\frac{\partial F}{\partial u}\right|_u (v), p \right\rangle$$

for any v and p. Here,

$$F(u) = -\mu \Delta u + \vec{\beta} \cdot \nabla u - f$$

is the functional related to the state problem, and

$$\left.\frac{\partial F}{\partial u}\right|_u (v) = F(v) = -\mu \Delta v + \vec{\beta} \cdot \nabla v$$

is the Fréchet operator of F applied to v. Then, we have

$$\int_\Omega p \left.\frac{\partial F}{\partial u}\right|_u^* (v) = \int_\Omega (-\mu \Delta v + \vec{\beta} \cdot \nabla v) p$$

Considering that

$$\nabla \cdot (\phi \nabla \psi) = \phi \Delta \psi + \nabla \phi \cdot \nabla \psi$$

$$\nabla \cdot (\psi \vec{A}) = \psi \nabla \cdot \vec{A} + \vec{A} \cdot \nabla \psi$$

$$\int_\Omega \nabla \cdot \vec{A} = \int_{\delta\Omega} \vec{A} \cdot \hat{n}$$

and assuming that $p(\delta\Omega) = 0$, it is possible to show that

$$\int_\Omega (-\mu \Delta v + \vec{\beta} \cdot \nabla v) p = \int_\Omega v(-\mu \Delta p - \vec{\beta} \cdot \nabla p)$$

Therefore, the adjoint operator is

$$\left.\frac{\partial F}{\partial u}\right|_u^* (p) = -\mu \Delta p - \vec{\beta} \cdot \nabla p$$

Adjoint problem Then, the adjoint problem requires to find p such that

$$\left\langle \left.\frac{\partial F}{\partial u}\right|_u^* (p), v \right\rangle = \left.\frac{\partial \mathcal{J}}{\partial u}\right|_u (v)$$

for all v. Given the definition of $\mathcal{J}$, we have

$$\left.\frac{\partial \mathcal{J}}{\partial u}\right|_u (v) = \int_\Omega (u-d)v$$

which gives the weak formulation of the adjoint problem

$$\int_\Omega (-\mu\Delta p - \vec{\beta}\cdot\nabla p)v = \int_\Omega (u-d)v$$

since this must hold true for any v, this is equivalent to the strong form

$$\begin{aligned} -\mu\Delta p - \vec{\beta}\cdot\nabla p &= u-d \qquad && x\in\Omega \\ p(x) &= 0 && x\in\delta\Omega \end{aligned}$$

which is the problem to be solved in order to find p.

Partial derivatives To compute the partials, we use the chain rule. For instance, working with β_x we have

$$\frac{D\mathcal{J}}{D\beta_x} = \frac{\partial \mathcal{J}}{\partial u}\left(\frac{\partial u}{\partial \beta_x}\right) + \frac{\partial \mathcal{J}}{\partial \beta_x}$$

Note that

$$\frac{\partial \mathcal{J}}{\partial \beta_x} = 0$$

since there is no explicit dependency of $\mathcal{J}$ on β_x. This will not be true if some terms are added for a regularazing effect. Going back, considering first the adjoint problem, and then the adjoint definition

$$\frac{D\mathcal{J}}{D\beta_x} = \frac{\partial \mathcal{J}}{\partial u}\left(\frac{\partial u}{\partial \beta_x}\right) = \left\langle \left.\frac{\partial F}{\partial u}\right|_u^* (p), \frac{\partial u}{\partial \beta_x} \right\rangle = \left\langle p, \left.\frac{\partial F}{\partial u}\right|_u \left(\frac{\partial u}{\partial \beta_x}\right) \right\rangle$$

Note that since

$$F(u) = 0$$

by chain rule, it is true that

$$\frac{\partial F}{\partial u}\left(\frac{\partial u}{\partial \beta_x}\right) + \frac{\partial F}{\partial \beta_x} = 0$$

Therefore, the partial derivative is given by

$$\frac{D\mathcal{J}}{D\beta_x} = \left\langle p, \left.\frac{\partial F}{\partial u}\right|_u \left(\frac{\partial u}{\partial \beta_x}\right) \right\rangle = \left\langle p, -\frac{\partial F}{\partial \beta_x} \right\rangle$$

Finally, considering F, this last term is

$$\frac{\partial F}{\partial \beta_x} = \frac{\partial u}{\partial x}$$

A similar procedure can be followed for β_y and μ.

2.4.4 Summary

To optimize $\mathcal{J}$, we need its gradient $\nabla\mathcal{J}$ evaluated at any value of the parameters (μ, β_x, β_y). To calculate it, we follow these steps

1. Given values μ, β_x, and β_y, find u by solving the state problem

$$-\mu\Delta u + \vec{\beta} \cdot \nabla u = f$$

$$u(\delta\Omega) = 0$$

This is done using FEniCS.

2. Using the same FEniCS solver, find p by solving the adjoint problem

$$-\mu\Delta p - \vec{\beta} \cdot \nabla p = u - d$$

$$p(\delta\Omega) = 0$$

3. Compute the gradient of $\mathcal{J}$ with the formulas

$$\frac{D\mathcal{J}}{D\mu} = \int_\Omega p\Delta u \tag{2.20}$$

$$\frac{D\mathcal{J}}{D\beta_x} = -\int_\Omega p u_x \tag{2.21}$$

$$\frac{D\mathcal{J}}{D\beta_y} = -\int_\Omega p u_y \tag{2.22}$$

The integration is done with FEniCS.

2.5 Numerical Hessian

The DD-CID algorithm requires the capacity to evaluate $\mathcal{J}$, its gradient, $\mathcal{J}$, and its Hessian, $H_\mathcal{J}$ for any value of the parameters. The previous two sections describe how to compute the first two, and this section briefly discusses the computation of the Hessian. This is a matrix of second derivatives, and for our two test problems, they are given by,

$$H_\mathcal{J} = \begin{bmatrix} \mathcal{J}_{\mu,\mu} & \mathcal{J}_{\mu,\beta} \\ \mathcal{J}_{\beta,\mu} & \mathcal{J}_{\beta,\beta} \end{bmatrix}$$

for the one-variable problem, and

$$H_{\mathcal{J}} = \begin{bmatrix} \mathcal{J}_{\mu,\mu} & \mathcal{J}_{\mu,\beta_x} & \mathcal{J}_{\mu,\beta_y} \\ \mathcal{J}_{\beta_x,\mu} & \mathcal{J}_{\beta_x,\beta_x} & \mathcal{J}_{\beta_x,\beta_x} \\ \mathcal{J}_{\beta_y,\mu} & \mathcal{J}_{\beta_y,\beta_x} & \mathcal{J}_{\beta_y,\beta_y} \end{bmatrix}$$

Where the double subscript notation represents a second derivative with respect of the parameters, that is,

$$J_{\alpha_1,\alpha_2} = \frac{D}{D\alpha_2}\left(\frac{D\mathcal{J}}{D\alpha_1}\right)$$

This is done numerically, using a simple first-order finite difference formula to approximate derivative. The formula used for the approximation of the derivative of a function f evaluated at x is given by

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Therefore, the mixed derivative $J_{\mu,\beta}$ evaluated at a point (m, b) is

$$J_{\mu,\beta}(m,b) \approx \frac{D\mathcal{J}/D\mu(m, b+h) - D\mathcal{J}/D\mu(m,b)}{h}$$

where the derivative $D\mathcal{J}/D\mu$ is known through the computation of the gradient of $\mathcal{J}$. The rest of second derivatives are computed in a similar way.

Chapter 3

Results and Conclusions

3.1 The coding environment: FEniCS

The previous chapter presented the procedure for computing the gradient of the misfit functional, ∇J. We learned that solving partial differential equations is a fundamental of this process. In general, such solutions can not be found analytically, and numerical methods are often required. *FEniCS* was used for this task, particularly for the multi-dimensional problem. FEniCS is a computer platform for solving partial differential equations through finite element methods [ABH+15]. The following is the code written in Python and shows how FEniCS was used to solve both the state and the adjoint equations, to ultimately compute ∇J for the problem in two dimensions.

```
# Solves the AD equation, in 2 dimensions, computes J and its
    gradient
#
# Depending on 'option', a different value is returned:
#
#    option == 'sol':   Returns the numerical solution
#    option == 'J':     Returns the value J, compared to the
    reference fuction
#    option == 'JGrad': Returns the gradient of J
```

```
#

def solver_gradient(mu, betax, betay, RHS, option):

    # Create mesh and define function space
    mesh = UnitSquareMesh(12, 12)
    V = FunctionSpace(mesh, 'P', 2)

    # Define physical constants
    mu = Constant(mu)
    beta = Constant((betax,betay))

    # Define Boundary Condition
    u_D = Constant(0.0) #<-- homogeneous BC


    def boundary(x, on_boundary):
        return on_boundary

    bc = DirichletBC(V, u_D, boundary)

    # Define variational problem
    u = TrialFunction(V)
    v = TestFunction(V)
    f = RHS

    # Bilinear form and functional
    a = mu*dot(nabla_grad(u), nabla_grad(v))*dx + dot(beta,
   nabla_grad(u))*v*dx
    L = f*v*dx

    # Compute solution
    # solver_parameters={"linear_solver": "cg", "preconditioner": "
   ilu"})
    u = Function(V)
    solve(a==L, u, bc)
    info(parameters,True)

    if option == 'sol': # Returns the numerical solution

        return u

    elif option == 'J': # Returns the value J

        # Get the reference as defined in get_reference() function
        u_ref = get_reference()

        # Compute the value of J
        J = ((l2_error)**2)/2
        return J

    elif option == 'JGrad': # Computes the gradient of J through the
    adjoint

        # Get the reference as defined in get_reference() function
        u_ref = get_reference()

        ########## Solve the adjoint problem ##########

        # Define variational problem
        p = TrialFunction(V)
        v = TestFunction(V)

        # Change the RHS
        f_new = u - u_ref

        # Bilinear form and functional
        a = mu*dot(nabla_grad(p), nabla_grad(v))*dx - dot(beta,
   nabla_grad(p))*v*dx
        L = f_new*v*dx

        # Compute solution
        p = Function(V)
        solve(a==L, p, bc)
        info(parameters,True)

        ########## Numerical Integration ##########

        # First partial derivative: with respect to mu
        J_mu = assemble(p*div(grad(u))*dx)

        # Second partial derivative: with respect to beta_x
        J_betax = -1*assemble(p*nabla_grad(u)[0]*dx)

        # Third partial derivative: with respect to beta_x
        J_betay = -1*assemble(p*nabla_grad(u)[1]*dx)

        ########## Gradient ##########
        G = np.array([J_mu, J_betax, J_betay])

        return G
```

3.2 From MATLAB to Python

The DD-CID algorithm was originally implemented in MATLAB by its authors. For this thesis, a Python version was developed so that it could be integrated with the

FEniCS environment. The translation was made manually line by line, and all of its composing functions were individually tested so that the results were similar to those of the original version. The final code was tested against the following functions:

- **Simple test function.** This function is taken from [DMZ20]. It is defined $f : \mathbb{R}^2 \to \mathbb{R}$ by

$$f(x_1, x_2) = (x_1^2 - 1)^2 + (x_1^2 + x_2 - 1)^2 \tag{3.1}$$

  It has two local minima at $(-1, 0)$, $(1, 0)$, and a saddle point at $(0, 1)$.

  The following table contains the global minima found using the Python version.

x_1	x_2	$f(x_1, x_2)$
-1.0	-6.6×10^{-12}	4.6×10^{-14}
1.0	-9.6×10^{-10}	3.00×10^{-23}

Table 3.1: Global minima - simple test function.

- **Rosenbrok function.** This function is taken from [JY13]. It is defined $f : \mathbb{R}^d \to \mathbb{R}$ with d a natural number. It is given by

$$f(x) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right] \tag{3.2}$$

  Its global minimum is $f(x) = 0$ at $x = (1, \ldots, 1)$. Taking $d = 3$, the following table contains the global minimum found using the Python version.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1.0	1.0	1.0	6.8×10^{-18}

Table 3.2: Global minimimum - Rosenbrok.

- **Boggs system.** This function is taken from [DMZ20]. It is defined $f : \mathbb{R}^2 \to \mathbb{R}$ by

$$f(x_1, x_2) = \frac{1}{2} S^T S \tag{3.3}$$

where

$$S(x_1, x_2) = \begin{bmatrix} x_1^2 - x_2 + 1 \\ x_1 - \cos(\pi x_2/2) \end{bmatrix} \tag{3.4}$$

It has 3 global minima at $(-1, 2)$, $(0, 1)$, and $(-\sqrt{2}/2, 3/2)$. The following table contains the global minima found using the Python version.

x_1	x_2	$f(x_1, x_2)$
-0.71	1.5	2.88×10^{-20}
-1.85×10^{-6}	1.00	5.82×10^{-13}
-1.00	2.00	2.00×10^{-14}

Table 3.3: Global minima - Boggs system with the Python version of DD-CID.

- **Shubert function.** This function is taken from [JY13]. It is defined $f : \mathbb{R}^2 \to \mathbb{R}$ by

$$f(x_1, x_2) = \left(\sum_{i=1}^{5} icos[(i+1)x_1 + i] \right) \left(\sum_{i=1}^{5} icos[(i+1)x_2 + i] \right) \tag{3.5}$$

This function has many critical points, including many global minima and saddle points. However, the lowest value achieved by the function is $f(x_{min}) = -186.7309$.

x_1	x_2	$f(x_1, x_2)$
-7.08	-7.71	-186.73

Table 3.4: Global minimum - Shubert function with the Python version of DD-CID.

3.3 Visualization of the functionals

The following are the plots of the two functionals.

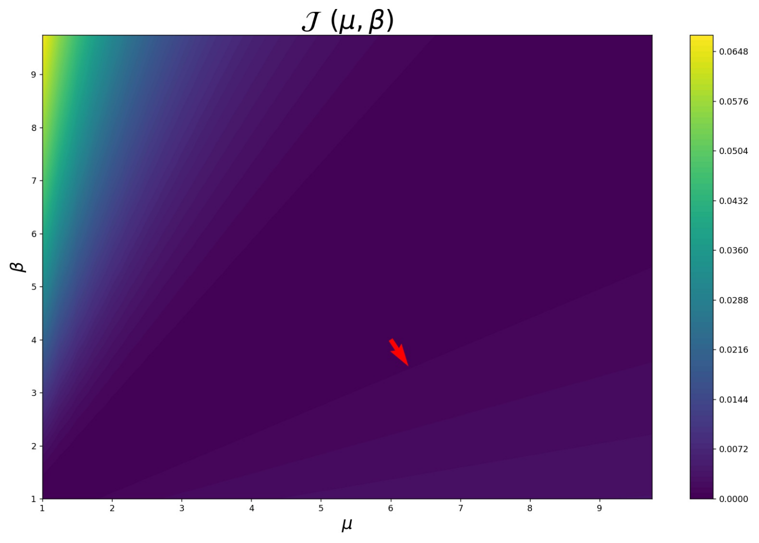

Figure 3.1: J for the one-dimensional problem.

Note that the global minima appears to be in line $\mu = \beta$. The gradient at the point $(6, 4)$ is shown for illustrative purposes.

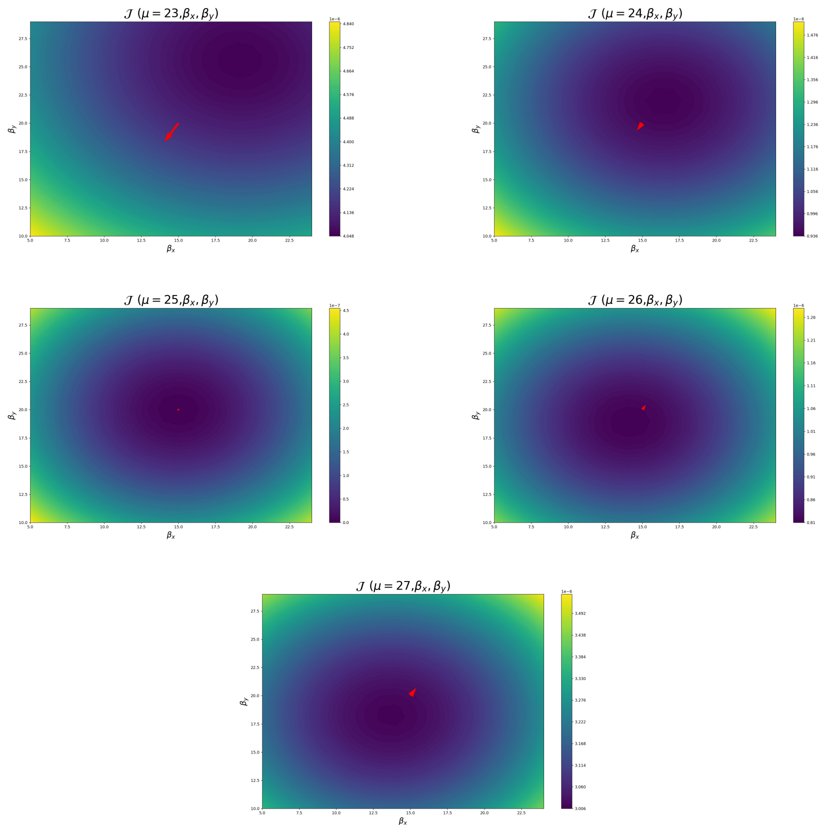

Figure 3.2: J for the multi-dimensional problem.

Note that the global minimum appears to be in line $(25, 15, 20)$. The gradient at the point $(\mu, 15, 20)$ is shown for illustrative purposes.

3.4 Results

The DD-CID algorithm was tested against the two problems described in Chapter 3. Here we present the results with $N = 5$ iterations of the main loop of the optimization algorithm. The two following tables contain the local minima found in each case. All of these are also global minima, given the nature of the two test problems. These results are precisely what was expected of the algorithm: in the one-variable case, different global minima are found along the line $\mu = \beta$, in the multi-variable case, the unique global minimum is found at $\mu = 25$, $\beta_x = 15$, and $\beta_y = 20$.

μ	β	$J(\mu, \beta)$
17.4	17.3	7.2×10^{-21}
27.9	27.9	1.5×10^{-22}
20.6	20.6	5.9×10^{-19}
28.2	28.2	7.9×10^{-17}
31.2	31.2	3.1×10^{-16}

Table 3.5: Global minima found by the DD-CID algorithm for the one-variable test problem. Initial point $\mu = 18.1$, $\beta = 16.6$

μ	β_x	β_y	$J(\mu, \beta_x, \beta_y)$
25.0	15.0	20.0	4.39×10^{-11}

Table 3.6: Global minimum found by the DD-CID algorithm for the multi-variable test problem. Initial point $\mu = 19.1$, $\beta_x = 9.4$, and $\beta_y = 15.4$

3.5 Conclusions

This work achieves two main objectives. First, it provides a working Python version of the DD-CID algorithm. This is relevant as Python is one the most popular programming languages nowadays. We hope that future work finds a Python version of the algorithm useful, by perhaps allowing its integration with tools only available in this language. This precisely leads to the second objective: testing the algorithm in a different setting. While DD-CID algorithm was previously shown to be effective for a variety of objective functions (see the original paper [DMZ20]), it had never been used in a parameter-estimation procedure like the one discussed in this work. Specifically, through the two test problems explored in the previous chapter, it was shown that the algorithm successfully optimized the the "mismatch functionals" that allowed the estimation. In this way, this thesis provides further evidence of the strength of the DD-CID algorithm as a general-purpose optimization method.

The most attractive feature of the DD-CID algorithm in the context of this work was its ability to determine multiple minima. Most parameter-estimation procedures that require an optimization step often resort to local algorithms such as Steepest Descent or Newton's Method. While these are successful, they are nonetheless local in the sense that they might not detect other minima that the function to be optimized may have. In other words, since the optimization problem does not have have a unique solution in general, a global optimization algorithm is highly recommended. This is was precisely exemplified by the one-variable test problem in the previous chapter, in which an infinite number of parameters corresponded to the reference function. The success of DD-CID algorithm in identifying these points allows us conjecture that it may be successful in real-world parameter-estimation problems. For instance, in [YV15], data assimilation is used to approximate cardiac conductivities of heart tissue. The authors point out that the uniqueness of the parameters was not guaranteed. Therefore, the DDCID algorithm could help us ensure the determination of the global minimum.

Bibliography

[ABH+15] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[APS+22] Shikha Agnihotry, Rajesh Kumar Pathak, Dev Bukhsh Singh, Apoorv Tiwari, and Imran Hussain. Protein structure prediction. In *Bioinformatics*, pages 177–188. Elsevier, 2022.

[BDPV14] Luca Bertagna, Marta D'Elia, Mauro Perego, and Alessandro Veneziani. Data assimilation in cardiovascular fluid–structure interaction problems: an introduction. In *Fluid-structure interaction and biomedical applications*, pages 395–481. Springer, 2014.

[DMZ20] Luca Dieci, Manuela Manetta, and Haomin Zhou. Double descent and intermittent color diffusion for landscape exploration. *Numerical Algorithms*, 85(1):145–169, 2020.

[Gun02] Max D Gunzburger. *Perspectives in flow control and optimization*. SIAM, 2002.

[JY13] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[Nav09] Ionel M Navon. Data assimilation for numerical weather prediction: a review. *Data assimilation for atmospheric, oceanic and hydrologic applications*, pages 21–65, 2009.

[YV15] Huanhuan Yang and Alessandro Veneziani. Estimation of cardiac conductivities in ventricular tissue by a variational approach. *Inverse Problems*, 31(11):115001, 2015.