Contents

Chapter 1

Introduction

In today's world high performance computing is an essential tool for scientists in many fields such as engineering, computational physics and chemistry, bioinformatics or weather forecasting. Many problems that arise in these areas are so computationally costly, that they can not be solved efficiently or at all on a single processor. Instead we solve or accelerate the solution of such problems by running them on large-scale clusters of multiple processes in parallel. One of the main issues with parallel computing is that most numerical solvers are sequentially formulated, and the work of translating these algorithms into a parallel framework can often be time and effort intensive.

One class of large-scale problems suited for parallelization, that frequently occurs both in science and engineering, are time dependent partial differential equations (PDEs). The traditional approach to implementing parallel solvers for such problems is to restrict the parallel computations to operations in spatial dimension at each time step, while the time-integration is done sequentially. Letting the implementation be serial in temporal direction is the most intuitive way of parallelizing time dependent PDEs, since evolving an equation in time is a naturally sequential process A lot of work has also been done on parallel solvers for spatially dicretized problems, meaning that methods and strategies for parallelization in space already are developed and tested. In all problems the spatially parallel solvers will eventually encounter upper bounds on achievable speedup when the number of available computing cores are high. Introducing parallelism in temporal direction is a way of increasing the speedup beyond this bound. It is therefore desirable to develop solvers for time dependent PDEs that are parallel in time.

There exists multiple methods for parallel in time solvers of evolution equations. The most famous and most developed of these parallel in time methods is the so called Parareal method introduced in [31]. The parallelism of Parareal is re-

stricted to the temporal dimension, and can therefore be used to parallelize both time dependent PDEs and ordinary differential equations (ODEs). It shares this feature with the related multiple shooting methods [6, 42], while waveform relaxation methods [19, 29] and multigrid methods [25, 27, 33] achieves parallelism in time by parallelizing in both space and time simultaneously. In this thesis we will restrict ourself to Parareal, and the other methods will not be touched upon any further.

The topic of this thesis is a subject closely related to time dependent differential equations (DE) namely optimization with time dependent DE constraints. Optimization with DE constraints are minimization problems of the following form:

$$\min_{y,v} J(y,v) \quad \text{subject to } E(y,v) = 0. \tag{1.1}$$

The functional J that we want to minimize is usually referred to as the objective function. $E(y,v) = 0$ represents the differential equation constraints, and is called the state equation. The state equations is solved for the state y, while the control v are parameters of the equation. The goal of the control problem (1.1) is to find a pairing $(\bar{y}, \bar{v})$ that minimize the objective function, but also satisfy the constraints set up by the state equation $E(\bar{y}, \bar{v}) = 0$. Optimization problems with PDE constraints have many applications. Some examples are: Optimal control ,variational data assimilation, optimal design and more. For further details on optimization we refer to [26].

Different strategies for numerically solving the optimal control problem (1.1) exists. One alternative is to set up and solve the optimality system stemming from the Lagrangian function associated with problem (1.1). In this thesis we will not use this approach, but instead reduce the constrained problem (1.1) into an unconstrained problem of type (1.2), and then solve this new problem using techniques from unconstrained optimization.

$$\min_{v} J(y(v), v). \tag{1.2}$$

Choosing this approach will obviously limit us to optimization problems where this reduction is possible. The process of finding the minimizer of problem (1.2) involves solving multiple differential equations. The computational cost of solving these equations will dominate the overall computational cost of any optimal control solver based on the reduction approach. Parallelization of problems of type (1.2) are therefore connected to the parallelization of differential equations. In this thesis we will develop and investigate a Parareal-based parallel in time framework for optimal control problems with time dependent DE constraints. To achieve this,

we will use the same strategy as in [37], meaning that we enforce the dependency between decomposed intervals by altering the objective function. The Parareal algorithm is then applied as a preconditioner for the optimization algorithm. In [37] the parallelization of time dependent optimal control problems is done by applying the Parareal preconditioner to the steepest descent method. We will instead test out an implementation where the same Parareal preconditioner is used in combination with the BFGS method. The algorithm that we present in this thesis is applicable to optimal control problems with ODE and PDE constraints, and for PDE constraints it can also be combined with spatial parallelization. For simplicity we will restrict the example problems to ordinary differential equation constraints.

1.1 Summary

The overall goal of this thesis is to establish a parallel in time algorithm for solving optimal control problems with time dependent DE constraints. The structure of the work done can roughly divided into two parts:

1. Background and presentation of algorithm
2. Verification and experiments

The bulk of the thesis is found in the first part, where we present and motivate a parallel framework for parallization of control problems. In chapter 2 we give a short literature review of previous work done on the Parareal algorithm, its theory and its application, emphasizing possible extensions to optimal control. In chapter 3 we look at general theory for optimal control with DE constraints. Among other things the adjoint approach to gradient evaluation is presented. Chapter 3 also includes one section on optimization algorithms and one on finite difference discretizations of ODEs. A more detailed, but still shallow presentation of the Parareal algorithm is found in chapter 4. Of special importance in this presentation, is the section on the alternative algebraic formulation of Parareal, since this formulation is used later in chapter 5 to derive the Parareal based preconditioner.

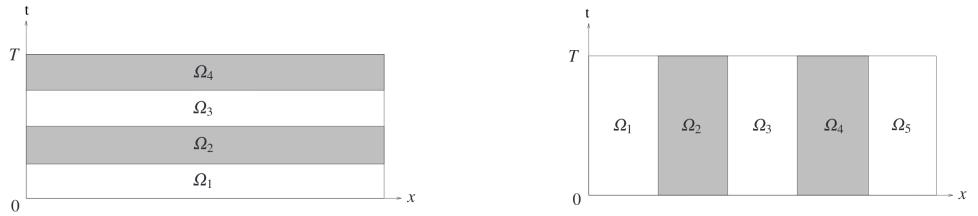
How we translate the solution process of time dependent optimal control problems into a parallel framework, is detailed in 5. Here we explain how to decompose the time domain of control problems, and how we can use the penalty method to enforce the continuity constraints that arises when decomposing in time. The rest of chapter 5 is dedicated to the presentation and derivation of the Parearal preconditioner. Since we want to use this preconditioner in combination with the BFGS optimization algorithm, we also need to check if the proposed preconditioner possesses the necessary mathematical properties for this to be possible.

The second part of the thesis deals with implementation, testing and verification of the algorithm. In chapter 6 we explain how we discretize the decomposed time domain and the non-penalized and penalized objective function for an example problem. In chapter 7 the discetized objective function and its gradient from chapter 6 is verified using the Taylor test. In the second part of chapter 6 we also explain how we implement the objective function and gradient evaluation in parallel using the message passing interface (MPI), with special attention on communication between processes and how this communication affects the speedup. The speedup of our implementation for function and gradient evaluation is also verified against the theoretical optimal speedup in chapter 7. Chapter 7 also demonstrates how the solution of the discretized control problem converges to the exact solution, and the consistency of the penalty framework presented in chapter 5. Chapter 8 contains the results of experiments done using the method from 5. The main focus of these results are the speedup this parallel algorithm produces. We measure speedup both in wall clock time and in a measure representing ideal speedup. This ideal speedup is based on the number of objective function and gradient evaluations done by the sequential and parallel algorithm.

Chapter 2

Literature review

As mentioned in [18] the Parareal algorithm is not the first attempt to parallelize the solution of time dependent differential equation in temporal direction, since Nievergelt already in 1964 proposed a procedure in [42] that eventually led to the so called multiple shooting methods. In [18] the authors also explain why the Parareal algorithm can be characterized as both a multiple shooting method and a multigrid method. A historic overview of the development of parallel in time algorithms can be found in [16]. Here we can also find presentations for the different strategies for parallelizing time dependent differential equations. One such strategy are the already mentioned multiple shooting methods [6, 42], which also include the Parareal algorithm. What characterizes such methods is that they only decompose the time domain. This separates the multiple shooting methods from waveform relaxation methods [19, 29], where the spatial domain is decomposed through time. The difference in these decomposition techniques is illustrated in figure 2.1. Other strategies presented are multigrid [25, 27, 33] methods and direct solvers in space-time [24, 35, 41].



(a) Multiple shooting decomposition (b) Waveform relaxation decomposition

Figure 2.1: Different decomposition techniques for parallel in time algorithms. (a) shows the strictly temporal decomposition of multiple shooting methods. In (b) the decomposition is done spatially through time. Image source: [16].

The Parareal algorithm was introduced by Lions, Maday and Turinici in [31] as a way to solve differential evolution equations $f(y(t), t) = 0$ in parallel. This is done by combining a coarse (but fast) and fine (but slow) numerical scheme for discretization in time. To introduce parallelism we first decompose the time domain $I = [0, T]$ into N subintervals $I_i = [T_{i-1}, T_i]$. This gives us N equations $f_i(y_i(t), t) = 0$ defined on each interval I_i.

The first step of the Parareal algorithm is to solve $f(y(t), t) = 0$ sequentially on the entire interval using the coarse scheme. This gives us a solution $Y(t)$ defined on the entire interval. We can then use $\{\lambda_i^0 = Y(T_i)\}_{i=1}^{N-1}$ as initial conditions for the decomposed equations $f_{i+1}(y_{i+1}^0(t), t) = 0$. The second step is then to solve these equations in parallel using the fine scheme, which will result in one solution $y_i(t)$ on each interval I_i. The idea then, is to utilize the difference $S_i^0 = y_i^0(T_i) - \lambda_i^0$ between coarse and fine solution to repeat this process in an iteration. This is done by propagating the differences S_i^0 with the coarse solver, to update the initial conditions for each decomposed equation. These new initial conditions λ_i^1 can then be used to solve the decomposed equations $f_{i+1}(y_{i+1}^1(t), t) = 0$ in parallel with the fine solver. We can then define updated differences $S_i^1 = y_i^1(T_i) - \lambda_i^1$ and repeat the iteration until we are satisfied with the solution. The version of Parareal presented in [31] is most practical for use on linear equations. An alternative version of Parareal algorithm is found in [2], which is equivalent to the one in [31] for linear equations, but is easier applied to non-linear equations.

A lot of the work on the Parareal algorithm has been focused on establishing its stability and convergence properties. The stability results are found in [49], [34] and [4]. In [34,49] sufficient conditions for the stability of Parareal for autonomous differential equations (2.1) is derived.

$$\frac{\partial y}{\partial t} = \rho y, \quad y(0) = y_0, \quad \rho < 0 \tag{2.1}$$

while [4] presents more general stability results for parabolic equations. The stability of Parareal applied to hyperbolic equations is a more difficult question as is explained in [11]. The convergence of Parareal is studied in [31], [4], [17] and [18]. In [31] Lions, Maday and Turinici show that k iterations of the Parareal algorithm applied to equation (2.1) gives $\mathcal{O}(\Delta T^{k+1})$ order of accuracy if we use a coarse solver with order one accuracy and coarse time step ΔT. This result is extended in [4] to more general equations, and the order of accuracy is shown to be improved to $\mathcal{O}(\Delta T^{p(k+1)})$ when the coarse solver has order p. [17, 18] return to analysis of equation (2.1). Instead of looking at a fixed number of iterations k, Gander and Vandewalle show convergence properties for the Parareal algorithm as the iteration count increases. They derived superlinear convergence for bounded time intervals

and linear convergence for unbounded time intervals.

The Parareal algorithm has been applied to different equations, including on the Navier-Stokes equations [14], to molecular-dynamics simulations [2], to stochastic ordinary differential equations [3], to reservoir simulations [20], to fluid, structure and fluid-structure problems [13], or on the American put [5]. The success of applying the Parareal algorithm varies between the different problems. For example in [5] a simulated speedup of 6.25 is achieved on 50 decompositions, which translates to an efficiency of 12.5%. In [13], speedups between 4.0 and 8.2 are achieved on twenty cores for an unsteady flow model. This corresponds to an efficiency of $20\% - 41\%$. The parallel in time algorithm was less successful when applied to structure and fluid-structure dynamics, since the authors of [13] here experienced difficulties with stability. For certain problem parameters, stability issues are encountered in [14], however for other parameters the algorithm is stable, and a speedup between 6.0 and 19.7 for 32 cores is estimated. This estimation, that assumes zero parallel overhead, would yield efficiency between 18.75% and 61.56%.

Since the Parareal algorithm is an iterative procedure, a stopping criteria for when to terminate the iteration is required. This is studied in [30], where an error control mechanism for the Parareal algorithm is introduced to limit the number of Parareal iterations. The stopping criteria that the authors propose stops the algorithm when the difference between coarse and fine solution at the subinterval boundaries T_i are similar to the expected global error of the fine solver. One of many challenges associated with parallel computing is partitioning and load bearing. This issue also arises in the Parareal algorithm, where the difficulties originates from the following observation: After k iterations of the Parareal algorithm, the solution in the k first subintervals is equal to the fine solution, see figure 2.2. This means that after k iterations, the the k-th process becomes idle. How to tackle this issue is described in [1], where the authors also present a practical implementation of the Parareal algorithm.

The Parareal algorithm parallelizes the solution process of time dependent differential equations. In [37] Maday and Turinici extend Parareal, for optimal control problems with time dependent differential equation constraints. In particular the problem looked at in [37], is:

$$\min_{y,u} J(y,u) = \frac{1}{2}\int_0^T ||u(t)||_U^2 dt + \frac{\alpha}{2}||y(T) - y^T||^2,$$
$$\begin{cases} \frac{\partial y}{\partial t} + Ay = Bu \\ y(0) = y_0 \end{cases}$$

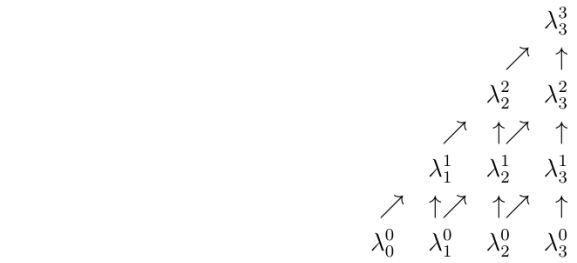


Figure 2.2: We see how the fine solution move from the initial condition at λ_0 to λ_3 through three iterations of the Parareal algorithm

The authors introduce parallelism in the same ways as for the differential equation case, by decomposing the time domain and equation. The continuity of the state equation between subintervals is enforced by adding a penalty term to the objective function J, that penalizes jumps in the solution of the state equation. This is based on the penalty method for constrained optimization described in [44]. In [37] they use quadratic penalty terms, which leads to the following modified objective function:

$$J_\mu(y, u, \lambda_1, .., \lambda_{N-1}) = J(y, u) + \frac{\mu}{2} \sum_{i=1}^{N-1} (y_i(T_i) - \lambda_i)^2 \tag{2.2}$$

The λ_i variables are called the virtual controls and are the initial conditions of the decomposed state equations $f(y_{i+1}(t), t) = 0$. Solving both the original and modified optimal control problems require us to repeatedly evaluate the objective function and its gradient. Every time we do this we need to solve either the state equation, or the state equation and its adjoint. Decomposing the time interval allows us to solve these equations in parallel, and if we solve the modified problem with a sufficiently large penalty, we will end up with the solution of the original problem. One does not necessarily need a coarse level to make this parallel framework produce a speedup. This is illustrated in [47], where the authors create a time-parallel algorithm for 4d variational data assimilation. The penalization of the objective function was done using the augmented Lagrangian approach, which is a variation of the penalization done in (2.2). The experiments conducted in [47] yielded limited success. Some speedup was achieved, however, the speedup was only attainable when using a parallel/sequential hybrid method, that first solved the penalized problem in parallel, for small penalty terms, and then used the parallel solution as an initial guess for the sequential algorithm.

In [37] the Parareal algorithm is reformulated as a preconditioner for the algebraic system that arises when we set $\lambda_i = y_i(T_i)$. Using this formulation the

authors derive a preconditioner for the optimization algorithm that solves the penalized optimal control problem. The preconditioner they propose involves both a backward and a forward solve of the linearised state equation with a coarse solver, and it is to be applied to the λ part of the gradient of J_μ. The motivation is that this Parareal based preconditioner could decrease the number of function and gradient evaluations needed for the optimization algorithm to converge, and the results in [37] do indeed look promising. In an experiment with 100 cores, the authors report a theoretical speedup of around 400, which is superlinear. They do however believe that this result is due to properties of the example they chose, and do not expect superlinerar speedup as a general rule.

The optimal control setting can also be used to modify the original Parareal algorithm. One example is [9], where the preconditioner for the optimal control problem from [37] is used in a modified Parareal algorithm to stabilize it for hyperbolic equations. The adjoint based Parareal algorithm is proposed in [46]. In this paper the authors address the bottleneck for speedup produced by having to repeatedly apply the coarse solver. This especially becomes a problem when the number of decompositions in time grows, while the problem size stays constant. The solution proposed in [46] is to only use the coarse solver once to get an initial guess for the intermediate initial conditions, and thereafter improve this initial guess by minimizing a functional of type (2.2) using an optimization algorithm. The optimization steps can be done completely in parallel, and the scalability of the adjoint based Parareal algorithm is therefore a lot better than the original.

In [36, 38] the authors derive a way to couple the Parareal algorithm with an optimization procedure for control of quantum systems. Like in [37] a penalty term is added to the objective function to handle the continuity constraints, but the optimization of the penalized functional is done in a slightly different way than in [37]. The approach taken in [36] is to minimize the penalized objective function using an alternating direction decent method. This means that the minimization of the functional of type 2.2 is done in two steps. First we minimize it for the virtual control $\{\lambda_i\}_{i=1}^{N-1}$, and then for the original control v. A Parareal step is incorporated into the minimization of the penalized objective function with respect to the virtual control variables.

We will in this thesis handle the DE constraints by moving them into the objective function, and therefore reducing the constrained optimization problems to unconstrained ones. An alternative to this strategy is the Lagrangian approach, where one first defines the Lagrangian function (2.3), and then derive the optimal-

ity system using the KKT-conditions.

$$\mathcal{L}(y, v, \lambda) = J(y, v) + \lambda E(y, v) \tag{2.3}$$

Some work has been done on trying to apply the Parareal algorithm to the solution process of the optimality system. For reference see: [8, 39, 45, 50].

Chapter 3

Optimal control with ODE constraints

In this chapter we present the basic mathematical background that the rest of the thesis will be based on. The chapter covers three different subjects. The first subject is on general theory of optimal control problems with DE constraints. The second subject is on finite difference discretization of differential equations and numerical integration, and the last subject deals with optimization algorithms. In addition to the general theory, we present an example optimal control problem with ODE constraints, that will be used throughout the rest of the thesis.

3.1 General optimal control problem

In this thesis we are only looking at optimization problems with time dependent differential equation (DE) constraints. This problem is only a part of the more general control problem, which we state in definition 1. Also included in definition 1 is the reducibility condition, which is a condition on the differential equation constraints.

Definition 1 (Optimization with DE constraints)**.** *Let Y, V, Z be Banach spaces, where Y, V also are reflexive. Given an objective function $J : Y \times V \to \mathbb{R}$ and an operator $E : Y \times V \to Z$, optimization with DE constraints then refers to the minimization problem on the following form:*

$$\min_{y \in Y, v \in V} J(y, v), \tag{3.1}$$

$$\text{Subject to: } E(y, v) = 0. \tag{3.2}$$

The differential equation $E(y, v) = 0$ is called the state equation, while the variables y and v are respectively known as the state and the control. If the following

condition holds:

$$\forall v \in V,\ \exists! y \in Y \ s.t.\ E(y,v) = 0, \tag{3.3}$$

we say the the optimization problem is reducible.

An alternative way of expressing the reducibility condition (3.3), is that for all controls $v \in V$ the differential equation $E(y,v) = 0$ is well posed. Optimization problems with ill posed state equations can both be solvable and interesting, but the methods introduced in this thesis does only work on reducible problems, and we will from this point always assume that the optimization problems we look at are reducible.

Before we explain how to solve optimization problems, we investigate under what conditions problem (3.1-3.2) even have a solution. To answer this question, we will write up a result from [26] concerning the existence and uniqueness of solution for linear quadratic optimization problems. This class of problems is less general than the problems from definition 1, and a more general existence results exist. To state this result however, would require the introduction of many complex concepts from functional analysis that go beyond the scope of this thesis. In addition the example problem that we will introduce in section 3.1.1 belongs to the linear quadratic class of optimization problems.

Theorem 1. *Assume that H, V are Hilbert spaces and that Y, Z are Banach spaces. Given vectors $q \in H$ and $g \in Z$, and bounded linear operators $A : Y \to Z$, $B : V \to Z$ and $Q : Y \to H$, we can define the linear quadratic optimization problems as follows:*

$$\begin{aligned} &\min_{y \in Y, v \in V} J(y,v) = \frac{1}{2}||Qy - q||_H^2 + \frac{\alpha}{2}||v||_V^2, \\ &\textit{Subject to: } Ay + Bv = g. \end{aligned}$$

If $\alpha > 0$, the above linear quadratic optimization problem has a unique solution pair $(y,v) \in Y \times V$.

Proof. See [26]. □

We now lay away the question of solvability, and continue with an explanation of how to solve problem (3.1-3.2). We start by revisiting the reducibility condition from definition 1. When the reducibility condition (3.3) holds the state can written as a function $y(v)$ implicitly defined through the state equation. Using $y(v)$ we are able to define the reduced optimization problem:

Definition 2 (Reduced problem)**.** *Consider the optimization problem from definition 1, and assume that reducibility condition (3.3) holds. We can then define the reduced objective function* $\hat{J} : V \to \mathbb{R}$ *as:*

$$\hat{J}(v) = J(y(v), v). \tag{3.4}$$

The reduced optimization problem is then defined as the unconstrained minimization problem:

$$\min_{v \in V} \hat{J}(v). \tag{3.5}$$

The problem (3.5) is called the reduced problem because we have moved the differential equation constraints into the functional. By doing this, we have transformed the constrained problem (3.1-3.2) into an unconstrained one (3.5), and we can therefore solve the reduced problem using tools from unconstrained optimization. To be able to use these tools, we will need a way to evaluate the gradient of the reduced objective function $\hat{J}$. There are several ways of doing this, but we will focus on the so called adjoint approach, which turns out to be the most computationally effective way to evaluate $\hat{J}'(v)$.

3.1.1 Example problem

To better understand the adjoint approach to gradient evaluation of the reduced objective function, we will define a simple optimal control problem with ODE constraints, so that we later can derive its adjoint equation and gradient. The problem will also be used to test and verify the implementation in chapter 7 and 8. In our example both the state y and the control v will be functions on an interval $[0, T]$. This allows us to define the objective function:

$$J(y, v) = \frac{1}{2} \int_0^T v(t)^2 dt + \frac{\alpha}{2}(y(T) - y^T)^2 \tag{3.6}$$

The state equation $E(y, v) = 0$ is a linear, first order equation with the control as a source term:

$$\begin{cases} y'(t) = ay(t) + v(t) \text{ for } \quad t \in (0, T), \\ y(0) = y_0. \end{cases} \tag{3.7}$$

The state equation of our optimal control problem is uniquely solvable for all integrable controls v, and has solution:

$$y(t) = e^{at}(C(y_0) + \int_0^t e^{-a\tau} v(\tau) d\tau)$$

This means that our example problem (3.6-3.7) is reducible. Since the state equation is linear, and since all terms in the objective function are quadratic, (3.6-3.7) is also an example of a linear quadratic optimization problem. Theorem 1 therefore guaranties a unique minimizer of problem (3.6-3.7).

3.2 The adjoint equation and the gradient

The usual way of finding the minimum (or maximum) value of a function $\hat{J}$, is to solve the equation $\hat{J}'(v) = 0$. Solving this equation usually requires us to be able to evaluate, or have an expression for the derivative of $\hat{J}$. There are different ways to evaluate the gradient of the reduced objective function $\hat{J}(v)$. We will here take the adjoint approach. This strategy leads to an expression for the gradient of the reduced objective function (3.4), which we state in proposition 1. The reason it is called the adjoint approach, is that the gradient $\hat{J}'(v)$ depends on the so called adjoint equation. The definition of this equation is found in Proposition 1. Proposition 1 also include conditions on the operators J and E from definition 1. These conditions involve the notion of Fréchet differentiability, which is a generalization of directional derivatives for operators on Banach spaces. For a more precise definition of Fréchet differentiability, we refer to [26].

Proposition 1 (Gradient and adjoint of the reduced objective function)**.** *Let $\hat{J}$ be the reduced objective function from definition 2, and assume that the state equation operator E and the objective function J are Fréchet differentiable. Assume also that the partial derivative $E_y(y, v) : Y \to Z$ of E with respect to y is a linear and continuously invertible operator. Then the gradient of $\hat{J}$ with respect to the control v is:*

$$\hat{J}'(v) = -E_v(y, v)^* p + J_v(y, v), \tag{3.8}$$

where p is the solution of the adjoint equation:

$$E_y(y, v)^* p = J_y(y, v). \tag{3.9}$$

Proof. If J and E are Fréchet differentiable and if E_y is continuously invertible, then the implicit function theorem ensures that $y(v)$ is continuously differentiable. For a more detailed discussion on the implicit function theorem, see [26]. To differentiate $\hat{J}(v) = J(y(v), v)$, we take the total derivative with respect to v D_v of the unreduced objective function:

$$\hat{J}'(v) = D_v J(y(v), v) = y'(v)^* J_y(y, v) + J_v(y, v).$$

The problematic term in the above expression, is $y'(v)^*$, since the function $y(v)$ is implicitly defined through E. We can however find an equation for $y'(v)^*$ if we the the total derivative of the state equation with respect to v.

$$\begin{aligned} D_v E(y(v), v) = 0 &\Rightarrow E_y(y, v)y'(v) = -E_v(y, v) \\ &\Rightarrow y'(v) = -E_y(y, v)^{-1}E_v(y, v) \\ &\Rightarrow y'(v)^* = -E_v(y, v)^* E_y(y, v)^{-*}. \end{aligned}$$

Instead of inserting $y'(v)^* = -E_v(y, v)^* E_y(y, v)^{-*}$ into our gradient expression, we define the adjoint equation as:

$$E_y(y, v)^* p = J_y(y, v).$$

This now allows us to write up the gradient as follows:

$$\begin{aligned} \hat{J}'(v) &= y'(v)^* J_y(y, v) + J_v(y, v) \\ &= -E_v(y, v)^* E_y(y, v)^{-*} J_y(y, v) + J_v(y, v) \\ &= -E_v(y, v)^* p + J_v(y, v). \end{aligned}$$

□

Expression (3.8) gives us recipe for evaluating the reduced objective function for a control variable $v \in V$. Typically this evaluation requires us to solving both the state and adjoint equation, and then inserting the adjoint into expression (3.8). To better illustrate how gradient evaluation works let us derive the adjoint equation and the gradient of the problem introduced in section 3.1.1.

3.2.1 Adjoint of the example problem

We want to derive the gradient of problem (3.6-3.7), and in order to do so, we need the adjoint equation of the problem, which we now state in the proposition below, followed by its derivation.

Proposition 2. *The adjoint equation of the problem (3.6-3.7) is:*

$$-p'(t) = ap(t) \tag{3.10}$$

$$p(T) = \alpha(y(T) - y^T) \tag{3.11}$$

Proof. Before we calculate the different terms used to derive the adjoint equation, we want to fit our ODE into an expression E. We do this by writing up the weak

formulation of the equation:

$$y \in L^2(0,T) \text{ such that}$$
$$L[y,\phi] = \int_0^T -y(t)\phi'(t) - ay(t)\phi(t)dt - y_0\phi(0) + y(T)\phi(T) - \int_0^T v(t)\phi(t) = 0$$
$$\forall\ \phi \in C^\infty((0,T))$$

To derive the adjoint we need E_y and J_y. For E_y we define $(\cdot,\cdot)$ to be the L^2 inner product over $(0,T)$. Since the weak formulation includes evaluation at $t=0$ and $t=T$, we define an operator δ_τ that represent function evaluation in an L^2 inner product setting. We do this in the following way: Let $\tau \in [0,T]$ then:

$$(v, \delta_\tau w) = (\delta_\tau v, w) = \int_0^T v(t)\delta_\tau w(t)dt = v(\tau)w(\tau)$$

Using the above notation, we can write E_y quite compactly as:

$$E_y = L_y[\cdot,\phi] = (\cdot, (-\frac{\partial}{\partial t} - a + \delta_T)\phi)$$

Let us be more thorough with J_y, which is the right hand side of the adjoint equation.

$$\begin{aligned}
J_y(y(v),v) &= \frac{\partial}{\partial y}(\frac{1}{2}\int_0^T v^2 dt + \frac{\alpha}{2}(y(T)-y^T)^2) \\
&= \frac{\partial}{\partial y}\frac{\alpha}{2}(y(T)-y^T)^2 \\
&= \frac{\partial}{\partial y}\frac{\alpha}{2}(\int_0^T \delta_T(y-y^T)dt)^2 \\
&= \alpha\delta_T\int_0^T \delta_T(y(t)-y^T)dt \\
&= \alpha\delta_T(y(T)-y^T)
\end{aligned}$$

We have $E_y = (\cdot, (-\frac{\partial}{\partial t} - a + \delta_T)\phi)$, but we want to find its adjoint E_y^*. Therefore let us explain what is ment by an adjoint on the context of the L^2 inner product $(\cdot,\cdot)$. Let $B : L^2(0,T) \to L^2(0,T)$ be a linear operator. Then the adjoint of B, B^* is an operator on $L^2(0,T)$, such that $\forall v, w \in L^2(0,T)$:

$$(Bv, w) = (v, B^*w).$$

The adjoint of the bilinear form $E_y = L_y$, would therefore be a bilinear form $L_y^* = E_y^*$ such that $\forall v, w \in L^2(0,T)$:

$$L_y[v,w] = L_y^*[w,v].$$

Therefore to derive the adjoint of E_y, we will insert two functions v and w into $L_y[v, w]$, and try to change the places of v and w.

$$\begin{aligned} E_y = L_y[v, w] &= \int_0^T -v(t)(w'(t) + aw(t))dt + v(T)w(T) \\ &= \int_0^T w(t)(v'(t) - av(t))dt + v(T)w(T) - v(T)w(T) + v(0)w(0) \\ &= \int_0^T w(t)(v'(t) - av(t))dt + v(0)w(0) \\ &= L_y^*[w, v] = E_y^* \end{aligned}$$

If we multiply J_y with a test function $\psi \in C^\infty((0,T))$ and set $L_y^*[p, \psi] = (J_y, \psi)$, we get the following equation: Find p such that:

$$\int_0^T p(t)\psi'(t) - ap(t)\psi(t)dt + p(0)\psi(0) = \alpha(y(T) - y^T)\psi(T) \quad \forall\ \psi \in C^\infty((0,T))$$

If we then do partial integration, the equation reads: Find p such that:

$$\int_0^T (-p'(t) - ap(t))\psi(t)dt + p(T)\psi(T) = \alpha(y(T) - y^T)\psi(T) \quad \forall\ \psi \in C^\infty((0,T))$$

Using this we get the strong formulation:

$$\begin{cases} -p'(t) = ap(t) \\ p(T) = \alpha(y(T) - y^T) \end{cases}$$

□

With the adjoint we can find the gradient of $\hat{J}$. Lets state the result first.

Proposition 3. *The gradient of the reduced objective function $\hat{J}$ with respect to v is*

$$\hat{J}'(v) = v + p. \tag{3.12}$$

Proof. Firstly we need J_v and E_v^*:

$$\begin{aligned} J_v &= v \\ E_v &= L_v[\cdot, \phi] = -(\cdot, \phi). \end{aligned}$$

Since $L_v[\cdot, \phi]$ is symmetric, $E_v^* = E_v$, and strongly formulated, $E_v = -1$. By inserting relevant terms into (3.8), we get the gradient:

$$\hat{J}'(v) = -E_v^* p + J_v \tag{3.13}$$

$$= p + v. \tag{3.14}$$

□

Evaluating the gradient of our example problem can now be boiled down to the following three steps:

1. Solve the state equation for y.
2. Use y to solve the adjoint equation for p.
3. Insert p and control v into gradient formula (3.12).

To see why the above procedure is computationally effective, let us compare it with the finite difference approach to evaluating the gradient. Using finite difference we can find an approximation of the directional derivative $(\hat{J}'(v), h)_V$ in direction $h \in V$, by choosing a small $\epsilon > 0$ and setting:

$$(\hat{J}'(v), h)_V \approx \frac{\hat{J}(v + \epsilon h) - \hat{J}(v)}{\epsilon} \tag{3.15}$$

To calculate the above expression, we need to evaluate the objective function at $v + \epsilon h$ and v. Since objective function evaluation requires the solution of the state equation, finding the directional derivative of $\hat{J}$ in a direction h involves solving two ODEs. We are however interested in the gradient of $\hat{J}$, not its directional derivatives. To find $\hat{J}'(v)$ we calculate (3.15) for all unit vectors in V. This assumes that V is a finite space, which is always true in the discrete case. If we now look at the discrete case and assume that $V = \mathbb{R}^n$ we can write up a recipe for finding $\hat{J}'(v)$ using finite difference. Let e_i denote the i-th unit vector of $\mathbb{R}^n$. $\hat{J}'(v)$ can then be found in the following way:

1. Evaluate $\hat{J}(v)$.
2. Evaluate $\hat{J}(v + \epsilon e_i)$ for $i = 1, ..., n$.
3. Set the i-th component of $\hat{J}'(v)$ to be $\dfrac{\hat{J}(v + \epsilon e_i) - \hat{J}(v)}{\epsilon}$.

To execute the above steps, we need to solve the state equation for $n + 1$ different control variables. In comparison finding $\hat{J}'(v)$ using the adjoint approach only requires us to solve the state and adjoint equations once, independently of the dimension of V. For finite difference the computational cost of one gradient evaluation therfore depends linearly on the number of components in the control variable v, while the computational cost of the adjoint approach is independent of the size of v.

3.2.2 Exact solution of the example problem

It turns out that we can find the exact solution of problem (3.6-3.6) by utilizing the adjoint equation (3.10-3.11) and the gradient of the reduced objective function

(3.14). Finding an exact solution to our example problem will be useful for us in chapter 7, where we will be testing and verifying different aspects of our algorithm. The derivation of the solution is based on two key observations. The first observation is a relation between the optimal control $\bar{v}$ and the adjoint p, which is a result from the trivial fact that $\hat{J}'(\bar{v}) = 0$ is a necessary condition for $\bar{v}$ being a minimizer of $\hat{J}$. Inserting expression (3.14) into $\hat{J}'(\bar{v}) = 0$ yields:

$$\bar{v}(t) = -p(t). \tag{3.16}$$

The second observation concerns the solution of the adjoint equation (3.10-3.11). Given a state $y(t)$, the solution of the adjoint equation is:

$$p(t) = \alpha(y(T) - y^T)e^{a(T-t)} = \omega e^{-at}. \tag{3.17}$$

Combining observation (3.16) with observation (3.17) suggests that a minimizer $\bar{v}$ of $\hat{J}$ should be on the form:

$$\bar{v}(t) = C_0 e^{-at}. \tag{3.18}$$

It turns out that plugging anstatz (3.18) into the state equation, and then using the resulting state to solve the adjoint equation makes us able to find the solution of our example problem. The solution is stated in proposition 4 followed by its derivation.

Proposition 4. *Assume $a \neq 0$ and $\alpha > 0$. Then the solution of optimal control problem(3.6-3.6) is the following function:*

$$\bar{v}(t) = \alpha \frac{e^{aT}(y^T - e^{aT}y_0)}{1 + \frac{\alpha e^{aT}}{2a}(e^{aT} - e^{-aT})} e^{-at} \tag{3.19}$$

Proof. We start the proof by writing up the state equation (3.7) with (3.18) as source term:

$$\begin{cases} y'(t) = ay(t) + C_0 e^{-at} \quad \text{for} \quad t \in (0, T), \\ y(0) = y_0. \end{cases}$$

This is a first order linear ODE with solution:

$$y(t) = y_0 e^{at} + \frac{C_0}{2a}(e^{at} - e^{-at}) \tag{3.20}$$

If we insert the state (3.20) into the formula for the adjoint (3.17), we can express the adjoint $p(t)$ in terms of the constant C_0:

$$p(t) = \alpha(y(T) - y^T)e^{a(T-t)} \tag{3.21}$$

$$= \alpha e^{aT}(y_0 e^{aT} + \frac{C_0}{2a}(e^{aT} - e^{-aT}) - y^T)e^{-at} \tag{3.22}$$

The last step is to plug $v(t) = C_0 e^{-at}$ and $p(t)$ from (3.22) into observation (3.16) and then solve for C_0:

$$\begin{aligned} v(t) = -p(t) &\iff C_0 e^{-at} = -\alpha e^{aT}(y_0 e^{aT} + \frac{C_0}{2a}(e^{aT} - e^{-aT}) - y^T)e^{-at} \\ &\iff C_0(1 + \frac{\alpha e^{aT}}{2a}(e^{aT} - e^{-aT})) = \alpha e^{aT}(y^T - y_0 e^{aT}) \\ &\iff C_0 = \alpha \frac{e^{aT}(y^T - e^{aT} y_0)}{1 + \frac{\alpha e^{aT}}{2a}(e^{aT} - e^{-aT})} \end{aligned}$$

Division by $(1+\frac{\alpha e^{aT}}{2a}(e^{aT}-e^{-aT}))$ is always allowed, since $\frac{1}{a}(e^{aT}-e^{-aT}) > 0, \forall a \neq 0$ and $\forall T > 0$. □

3.3 Numerical solution

To be able to solve optimal control problems numerically, we need to discretize the objective function and the state and adjoint equations. We are mainly interested in time dependent equations, and the standard way of discretizing ODEs or PDEs in temporal direction, is to use a finite difference method. Since the objective function includes an integral term, we also need methods for numerical integration. In this section we will only look at first order equations on the following form:

$$\begin{cases} \frac{\partial}{\partial t} y(t) = F(y(t), t), \quad t \in I \\ y(0) = y_0 \end{cases} \tag{3.23}$$

Both the state and adjoint equation of our example problem can be formulated as an equation on form (3.23), and understanding the numerics of (3.23) is therefore sufficient for the purposes of this thesis. Before we introduce numerical methods for solving ODEs and evaluating integrals, we need to explain how we discritize the time domain $I = [0, T]$. We do this by dividing I into n parts of length $\Delta t = \frac{T}{n}$, and then setting $t_k = k\Delta t$. This gives us a sequence $I_{\Delta t} = \{t_k\}_{k=0}^n$ as a discrete representation of the interval I. Numerically solving a differential equation for y on $I_{\Delta t}$ means that we try to find $y(t_k)$ for $k = 0, ..., n+1$. For the rest of this section we let the notation y_k denote evaluating the function y at time t_k.

3.3.1 Discretizing ODEs using finite difference

Finite difference is a tool for approximating derivatives of functions. When we have a discretized domain $I_{\Delta t}$ with time step Δt, the derivative of a function y at point t_k is approximated by:

$$\frac{\partial}{\partial t} y(t_k) \approx \frac{y_k - y_{k-1}}{\Delta t}. \tag{3.24}$$

By exploiting approximation (3.24) we can create methods for solving ODEs. This is done by relating y_k to neighbouring values y_j, $j \neq k$ through the ODE. The most simplistic examples of such finite difference methods are the explicit and implicit Euler methods. We write up these methods applied to (3.23) in definition 3 below.

Definition 3. *Explicit Euler applied to equation (3.23) means that for $k = 1, ..., n$ the value of y_k is determined by the following formula:*

$$y_k = y_{k-1} + \Delta t F(y_{k-1}, t_{k-1}). \tag{3.25}$$

If one instead uses implicit Euler the expression for y_k is:

$$y_k = y_{k-1} + \Delta t F(y_k, t_k). \tag{3.26}$$

By looking at expression (3.25) and (3.26) we see the origin of the names of the Euler methods. In the formula for implicit Euler, y_k appears on both sides of the equal sign, and is therefore implicitly defined. For the explicit Euler scheme y_k only appears on the left-hand side of expression (3.25), which means y_k is defined explicitly, and hence the name explicit Euler. Another thing to notice about the finite difference schemes in definition 3, is that they solve the equation forwardly. This means that given y at time t_K, we can use (3.25) and (3.26) to find y_j for $j > K$. The adjoint equation of optimal control problem with time dependent DE constraints is however solved backwards in time. We therefore need finite difference schemes for solving ODEs backwards. This is easily achieved by rearranging expression (3.25) and (3.26) in definition 3. A backwards solving explicit Euler scheme is found by adjusting the forward solving implicit Euler scheme, while a backwards implicit Euler method is derived by rearranging the forward explicit Euler formula. These modified backwards solving schemes are written up in definition 4.

Definition 4. *An explicit Euler finite difference scheme for equation (3.23) with initial condition at $t = T$ instead of $t = 0$ yields the following formula for y_k:*

$$y_k = y_{k+1} - \Delta t F(y_{k-1}, t_{k-1}). \tag{3.27}$$

If one instead uses implicit Euler the expression for y_k is:

$$y_k = y_{k+1} - \Delta t F(y_k, t_k). \tag{3.28}$$

We say that both the explicit and implicit Euler methods have an accuracy of order one. To explain what we mean by this, let us assume that we know that the function $\hat{y}$ solves equation (3.23) for a given F, and that $\hat{y}$ is sufficiently smooth. If we then use method (3.25) or (3.26) with some Δt to solve (3.23) numerically, there

exists a constant C such that the following error bound between $\hat{y}$ and numerical solution y holds:

$$\max_{k=0,\ldots,n} |y_k - \hat{y}(t_k)| \leq C\Delta t \tag{3.29}$$

A more accurate but still simple alternative to the Explicit and implicit Euler finite difference methods, is the so called Crank-Nicolson method [10]. We write up this method in a definition:

Definition 5. *The Crank-Nicolson finite difference scheme applied to equation (3.23) produces the following formula for y_k:*

$$y_k = y_{k-1} + \frac{\Delta t}{2}(F(y_k, t_k) + F(y_{k-1}, t_{k-1})). \tag{3.30}$$

In a setting where we are solving (3.23) backwards in time, the expression for y_k is changed to:

$$y_k = y_{k+1} - \frac{\Delta t}{2}(F(y_k, t_k) + F(y_{k-1}, t_{k-1})). \tag{3.31}$$

When comparing (3.30) with (3.25) and (3.26) we notice that the formula for y_k in the Crank-Nicolson method is simply the average between the formulas for y_k in the explicit and implicit Euler methods. We improve the accuracy by one order if we use Crank-Nicolson instead of the Euler methods. This means that the bound stated in (3.29) is improved to:

$$\max_{k=0,\ldots,n} |y_k - \hat{y}(t_k)| \leq C\Delta t^2 \tag{3.32}$$

Other more accurate finite difference schemes exist, but in this thesis we restrict the usage of finite difference methods to the ones presented in this section.

3.3.2 Numerical integration

In this subsection we present three simple methods for numerical integration. We need such methods since the objective function in our example problem (3.6) includes an integral. The methods that we present in definition 6 are called the left-hand rectangle rule, the right-hand rectangle rule and the trapezoid rule. Their names stem from the geometrical objects used to estimate the area under the function we want to integrate.

Definition 6. *We want to estimate the integral* $S = \int_0^T v(t)dt$ *numerically with a discretized time domain* $I_{\Delta t} = \{t_k\}_{k=0}^n$*. The left-hand rectangle rule approximates* S *using the following formula:*

$$S_l = \Delta t \sum_{k=0}^{n-1} v_k \tag{3.33}$$

A slightly different approach to estimating S *is the right-hand rectangle rule, defined by a formula similar to (3.33):*

$$S_r = \Delta t \sum_{k=1}^{n} v_k \tag{3.34}$$

A third way of approximating S *is the trapezoid rule:*

$$S_{trap} = \Delta t \frac{v_0 + v_n}{2} + \Delta t \sum_{k=1}^{n-1} v_k \tag{3.35}$$

The rectangle methods in definition 6 are of accuracy order one, while the trapezoid rule is of second order. It turns out that the above presented numerical methods are analogue to the three finite difference schemes stated in section 3.3.1. The left- and right-hand rectangle methods are related to the explicit and implicit Euler schemes, while the trapezoid rule is connected with Crank-Nicolson. When making numerical solvers for optimal control problems it therefore makes sense to discretize the differential equation and integral evaluation using analogue methods.

3.4 Optimization algorithms

Deriving and solving the adjoint equation gives us a way of evaluating the gradient of optimal control problems with ODE constraints. With the gradient we can solve our optimal control problems numerically by using an optimization algorithm. There exists many different optimization algorithms, but here we will only look at line search methods that are useful to us in this thesis. The methods we present are the steepest descent method and the related BFGS and L-BFGS methods.

3.4.1 Line search methods and steepest descent

Line search methods are algorithms used to solve problems of the type:

$$\min_x f(x), \ f : \mathbb{R}^n \longrightarrow \mathbb{R}$$

All line search methods are iterative methods that starts at an initial guess x^0 and generate a sequence $\{x^k\}$ that hopefully will converge to a solution. The k-th iteration in the algorithm can be described in the following way:

1. *Choose direction* $p_k \in \mathbb{R}^n$
2. *Choose step length* $\alpha_k \in \mathbb{R}$
3. *Set* $x^{k+1} = x^k + \alpha_k p_k$

If f is differentiable, a necessary condition for a point $x^* \in \mathbb{R}^n$ to be a minimizer of f, is that $\nabla f(x^*) = 0$. This optimality condition is used to create a stopping criteria for line search methods in the following way: Given a tolerance $\tau > 0$ and a norm $||\cdot||$ stop the line search iteration when

$$||\nabla f(x^k)|| < \tau \tag{3.36}$$

What separates different line search methods, is how one chooses descent direction p_k and step length α_k. Let us start with how to choose a good step length. There are several ways of doing this, but for our purposes the so called Wolfe conditions will suffice. The Wolfe conditions consists of two conditions on f, presented below:

$$\begin{aligned} f(x^k + \alpha_k p_k) &\leq f(x^k) + c_1 \alpha_k \nabla f(x^k) \cdot p_k \\ \nabla f(x^k + \alpha_k p_k) \cdot p_k &\geq c_2 \nabla f(x^k) \cdot p_k \end{aligned}$$

Here we use constants $0 < c_1 < c_2 < 1$. The first Wolfe condition ensures that the decrease in function value is proportional to both step length and direction. The second condition is that the gradient of f at $x^k + \alpha_k p_k$, should be less steep than at x^k, and therefore closer to fulfilling the optimality condition (3.36). If we can find a step length that satisfies these conditions we will use it. How to actually find a step length that satisfies the Wolfe conditions is quite involved, and we will therefore not go into this topic any further. Instead let us look into a couple of line search methods that will be used later in the thesis, starting with steepest descent.

The steepest descent method is a very simple line search method, where the step length p_k is set to the negative gradient direction at point x^k, i.e $p_k = -\nabla f(x^k)$. This gives us the following update for each iteration:

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k) \tag{3.37}$$

The problem with steepest descent is that it converges quite slowly. To understand why let us first write up a definition that characterizes convergence rates.

Definition 7. *We say that a sequence $\{x^k\}$ converges linearly to a limit L, if there exists $\epsilon \in (0,1)$ such that*

$$\lim_{k\to\infty} \frac{||x^{k+1} - L||}{||x^k - L||} = \epsilon.$$

If $\epsilon = 0$ we say that $\{x^k\}$ converges superlinearly to L, while $\epsilon = 1$ is characterized as sublinear convergence. Lastly we say that $\{x^k\}$ converges quadratically towards L, if

$$\lim_{k\to\infty} \frac{||x^{k+1} - L||}{||x^k - L||^2} = \epsilon.$$

With definition 7 in mind let us state a theorem from [44] that specifies the convergence rate of the steepest descent method.

Theorem 2. *Assume that $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is twice continuously differentiable, that the steepest decent method converge to a point x^*, and that the Hessian of f at this point, $\nabla^2 f(x^*)$ is positive definite. Then the following holds:*

$$f(x^{k+1}) - f(x^*) \leq (\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1})^2 (f(x^k) - f(x^*))$$

Here $\lambda_1 \leq \cdots \leq \lambda_n$ denotes the eigenvalues of $\nabla^2 f(x^*)$.

Proof. See [44]. □

The bound for $f(x^{k+1}) - f(x^*)$ given in theorem 2 corresponds to a linear convergence with $\epsilon = (\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1})^2$. For badly conditioned Hessians $\nabla^2 f(x^*)$, meaning $\lambda_n >> \lambda_1$, ϵ will approach one, and the convergence rate becomes almost sublinear. In general the linear convergence rate of steepest descent is considered poor, and we need improved algorithms to get faster convergence.

3.4.2 BFGS and L-BFGS

Since steepest descent has slow convergence, one usually uses faster line search methods to solve numerical optimization problems. One alternative is Newtons method. In Newtons method the search direction p_k is found by multiplying the inverse Hessian with the the negative gradient at x^k. This results in the following iteration:

$$x^{k+1} = x^k - \nabla^2 f(x^k)^{-1} \nabla f(x^k) \tag{3.38}$$

As we will see in theorem 3, the convergence of the Newton method relies on quite strict conditions on the Hessian $\nabla^2 f(x^k)$, which are not always satisfied. An

alternative to the Newton method is so called quasi-Newton methods. Instead of applying $\nabla^2 f(x^k)^{-1}$ to the negative gradient direction, such methods apply approximations of the inverse Hessian to $-\nabla f(x^k)$. The approximate Hessians are constructed for each x^k, using information from previous iterates. One well known quasi-Newton method is the BFGS method [7, 15, 22, 48]. In BFGS the inverse Hessian approximation is calculated by the following recursive formula:

$$H^{k+1} = (\mathbb{1} - \rho_k S_k \cdot Y_k) H^k (\mathbb{1} - \rho_k Y_k \cdot S_k) + S_k \cdot S_k, \tag{3.39}$$

$$S_k = x^{k+1} - x^k, \tag{3.40}$$

$$Y_k = \nabla f(x^{k+1}) - \nabla f(x^k), \tag{3.41}$$

$$\rho_k = \frac{1}{Y_k \cdot S_k}, \tag{3.42}$$

$$H^0 = \beta \mathbb{1}. \tag{3.43}$$

The above formula is designed in such a way, that H^k is symmetric positive definite. This gives us a requirement for the initial inverted Hessian approximation H^0, namely that it also needs to be symmetric positive definite. The usual choice however, is just identity or a multiple β of the identity, where the multiple reflects the scaling of the variables. Strategies of how to chose a scaling factor β is detailed in [32] and [21]. Each line search iteration for BFGS looks like:

$$x^{k+1} = x^k - \alpha H^k \nabla f(x^k) \tag{3.44}$$

In the BFGS method information from all previous iterations is used to create the inverse Hessian approximation for the new iteration. An alternative to this is to limit the number of iterations the recursive formula remembers to only the latest iterations. This variation of the BFGS method is called L-BFGS [43]. The length of the memory need to be chosen in advance, and the typical choice is 10. Two advantages L-BFGS has over BFGS is firstly that it requires less memory storage than BFGS. The second advantage is that limiting the memory of the inverse Hessian approximation accelerates the convergence of BFGS. This is demonstrated in [32] for several different optimization problems. The reason for the improved convergence, is that more recent iterates possess more relevant information for the current Hessian, and by emphasizing the more relevant information, we improve the approximation of the Hessian.

Convergence results for Newton and quasi-Newton methods

Both Newton and quasi-Newton methods converge faster than the steepest descent method. To show this we will include a couple of theorems from [44] concerning this topic. We start with a result on the convergence rate of Newtons method.

Theorem 3. *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, and that the Hessian $\nabla^2 f(x)$ is Lipschitz continuous in the neighbourhood of a solution x^* that satisfies $\nabla f(x^*) = 0$ and that $\nabla^2 f(x^*)$ is positive definite. Then the following holds for the Newton iteration 3.38:*

1. *If x^0 is suffichently close to x^*, the sequence of iterates converge to x^*.*
2. *The rate of convergence of $\{x^k\}$ is quadratic*
3. *The sequence of gradient norms $\{||\nabla f(x^k)||\}$ converges towards zero quadratically*

Proof. See [44]. □

The quadratic convergence of the Newton iteration is a big improvement in comparison with steepest descent, however theorem 3 also highlights one of the problems with the method. Since we need to invert $\nabla^2 f(x^k)$ to find the search direction at x^k, we need an initial x^0 sufficiently close to the actual solution for the iteration to even work. This problem does not arise in BFGS and L-BFGS, since the Hessian approximation is designed to be invertible. Unfortunately though, these quasi-Newton methods does not have the convergence properties of Newtons method, as the next result shows.

Theorem 4. *Assume $f : \mathbb{R}^n \to \mathbb{R}$ is three times differentiable. Consider then the quasi-Newton iteration $x^{k+1} = x^k - \alpha_k B_k^{-1} \nabla f(x^k)$, where B_k is an approximation of the Hessian along the search direction $p_k = -B_k^{-1} \nabla f(x^k)$, satisfying the condition:*

$$\lim_{K \to \infty} \frac{||(B_k - \nabla^2 f(x^k))p_k||}{||p_k||} = 0$$

If the sequence $\{x^k\}$ originating from the quasi-Newton iteration converges to a point x^*, where $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, the convergence is superlinear.

Proof. See [44]. □

Even though quasi-Newton methods do not posses the quadratic convergence of the Newton method, superlinear convergence is still a lot better than the linear convergence of steepest descent.