Optimal Control of PDEs solved by Fixed Point Iterations using Algorithmic Differentiation

by

Tim-Adrian Albring

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Computational Engineering Science

at RWTH Aachen University

Faculty of Mechanical Engineering

December 23, 2012

Thesis Supervisors:

Prof. Dr. Nicolas Gauger and Dipl.-Math. Max Sagebaum

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Aachen, December 23, 2012 _____

Tim-Adrian Albring

Contents

List of Figures

Chapter 1.

Introduction

Nowadays the efficiency of many technical systems has reached a point where it is not possible anymore to achieve improvements by manually varying some parameters. Sophisticated mathematical models are necessary to describe the physical characteristics which then in turn are used in optimization workflows to generate enhanced designs and other properties. Typically such models contain non-linear partial differential equations whose disretizations yield large non-linear systems of equations, and within an optimization framework these systems have to be solved multiple times. A common type of method for solving these systems are fixed-point iterations that give the possibility to obtain approximations of arbitrary accuracy. Large scale optimizations are often only feasible for rather low accuracies. Therefore we can ask the naturally arising question: How does the optimization algorithm behave if the solution of the describing system is only a low order approximation?

In the present work this problem is approached by a numerical investigation of an Optimal Control problem. This is essentially an optimization problem where a partial differential equation (PDE) is applied as a constraint. Representative for an actual physical problem a non-linear Poisson equation is used for this purpose. The discretization is performed using a *discontinous Galerkin* approach and the resulting non-linear system is then solved with *Newton's method*. For the optimization the *discrete Adjoint* approach has been implemented. The gradients required by this method are calculated via *Algorithmic Differentiation*. Of particular importance is the so-called adjoint variable which contains information about the sensitivity of the fixed-point solver with respect to its input variables. Four different methods are compared that differ in their use of information generated during the solution of the non-linear system. At the end results are presented that show how these methods perform for different accuracies of the fixed-point solver especially in cooperation with the optimization procedure.

Chapter 2.

Algorithmic Differentiation

In recent numerical applications in engineering and sciences the handling of gradients has become an important task. Especially for optimization problems, where gradient-based methods perform much better than gradient-free methods, the need for an efficient evaluation of gradients has grown in the past. Although the performance of supercomputers is increasing and finite difference (FD) methods may be affordable for smaller problems with few unknowns, for most of the problems with many unknowns it is still not an option.

Algorithmic Differentiation or also called *automatic differentiation* is a way to calculate the derivative of a function by means of the transformation of the underlying program which calculates the numerical values of this function. As distinguished from symbolic differentiation an explicit expression for the derivative is never formed. An advantage over FD is that no truncation errors are present, thus the numerical value can be determined up to machine accuracy. Furthermore it is possible to get the whole or part of the Jacobian with less effort.

2.1. Forward Propagation

Suppose we have a function $f \in C^1 : U \subset \mathbb{R}^n \to \mathbb{R}^m$ and the expression

$$y = f(x), \tag{2.1}$$

that is, $x \in U, y \in \mathbb{R}^m$. The function f can be decomposed in a sequence of l elementary functions φ_i and intermediate values v_i:

$$v_{i-n} = x_i, \qquad i = 1 \dots n \tag{2.2}$$

$$v_i = \varphi_i(v_j)_{j \prec i}, \qquad i = 1 \dots l \tag{2.3}$$

$$y_{m-i} = v_{l-i}, \qquad i = m-1 \dots 0 \tag{2.4}$$

where $j \prec i$ means that i depends on j. Applying the chain rule to equation (2.3) results in

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial}{\partial v_j} \varphi_i(u_i) \dot{v}_j \tag{2.5}$$

with the abbreviation $u_i = (v_j)_{j \prec i}$. For the derivative of the function f then holds

$$\dot{y}_{m-i} := \dot{v}_{l-i} \equiv \frac{\partial f(x)}{\partial x_{m-i}}, \quad i = m-1 \dots 0 \tag{2.6}$$

That means, as long as the derivatives of the elementary functions are known a straightforward evaluation yields the derivative of f. Note that for building the derivative of the evaluation trace (2.2) - (2.4) we need the derivatives of the values $v_{i-n}, i = 1 \dots n$ as an input, that is, values for $\dot{x}_i$. By formally applying the chain rule to the expression (2.1) we get

$$\dot{y} := \frac{df}{dx}(x) \cdot \dot{x} = \sum_{i=1}^{m} \frac{\partial f(x)}{\partial x_i} \dot{x}_i \tag{2.7}$$

Thus we can get the derivative $\frac{\partial f(x)}{\partial x_i} \in \mathbb{R}^m$ by setting $\dot{x}_i = 1$ and $\dot{x}_j = 0, j \neq i, j = 1, \dots, n$. If equations (2.5) and (2.6) are combined with the evaluation trace one gets the Tangent Recursion for General Evaluation Procedure as in [GW08]:

$$[v_{i-n}, \dot{v}_{i-n}] = [x_i, \dot{x}_i] \qquad i = 1 \dots n, \tag{2.8}$$

$$[v_i, \dot{v}_i] = [\varphi_i(v_j)_{j \prec i}, \sum_{j \prec i} \frac{\partial}{\partial v_j} \varphi_i(u_i) \dot{v}_j] \qquad i = 1 \dots l, \tag{2.9}$$

$$[y_{m-i}, \dot{y}_{m-i}] = [v_{l-i}, \dot{v}_{l-i}], \qquad i = m-1 \dots 0. \tag{2.10}$$

Example 1

Consider the function $g : \mathbb{R}^2 \to \mathbb{R}^2$ and the expression:

$$y = g(x_1, x_2) := \begin{pmatrix} \sin(x_1) + e^{x_2} \\ \frac{\sqrt{x_2}}{x_1} \end{pmatrix} \tag{2.11}$$

The evaluation procedure for this function looks like the following:

$$\begin{aligned}
v_{-1} &= x_1, & \dot{v}_{-1} &= \dot{x}_1, \\
v_0 &= x_2, & \dot{v}_0 &= \dot{x}_2, \\
v_1 &= \sin(v_{-1}), & \dot{v}_1 &= \cos(v_{-1})\dot{v}_{-1}, \\
v_2 &= e^{v_0}, & \dot{v}_2 &= e^{v_0}\dot{v}_0, \\
v_3 &= \sqrt{v_0}, & \dot{v}_3 &= \frac{1}{2\sqrt{v_0}}\dot{v}_0, \\
v_4 &= \frac{1}{v_{-1}}, & \dot{v}_4 &= -\frac{1}{v_{-1}^2}\dot{v}_{-1}, \\
v_5 &= v_1 + v_2, & \dot{v}_5 &= \dot{v}_1 + \dot{v}_2, \\
v_6 &= v_3 v_4, & \dot{v}_6 &= \dot{v}_3 v_4 + v_3 \dot{v}_4, \\
y_1 &= v_5, & \dot{y}_1 &= \dot{v}_5, \\
y_2 &= v_6, & \dot{y}_2 &= \dot{v}_6.
\end{aligned}$$

In order to evaluate the Jacobian $\frac{dg}{dx}(x_1, x_2) \in \mathbb{R}^2 \times \mathbb{R}^2$ at a point $(w_1, w_2)^T$ the evaluation trace has to be called with $(x_1, x_2)^T = (w_1, w_2)^T$ and $\dot{x} = e_i, i = 1{,}2$, where $e_i \in \mathbb{R}^2$ is the i-th cartesian basis vector. Thus, in general n calls of the evaluation trace are necessary to accumulate the full jacobian.

2.2. Reverse Propagation

For the forward mode the derivatives are propagated in the same direction as the corresponding elemental function values, i.e. we ask how a infinitesimal change in the input values propagates through the evaluation trace and affects the output. But we could also ask the other way round: How sensitive are the output values to a change in the input values? The latter is the basis for the reverse propagation. There are many ways in which the reverse mode can be introduced, but in general it can be thought of as the backward application of the chain rule.

A sophisticated derivation can be found in [GW08]. Here only the main results needed for the application are described.

For the application of the chain rule to the decomposition of the function f (equation (2.2) – (2.4)) it is useful to introduce the state transformation Φ_i for each elemental function φ_i:

$$\mathbf{v}_i = \Phi_i(\mathbf{v}_{i-1}), \quad \Phi_i : \mathbb{R}^{n+l} \to \mathbb{R}^{n+l} \tag{2.12}$$

with

$$\mathbf{v}_i := (v_{1-n}, \ldots, v_i, 0, \ldots, 0)^T \in \mathbb{R}^{n+l}$$

We can then write the expression (2.1) as the composition

$$y = Q_m \Phi_l(\Phi_{l-1}(\ldots(\Phi_1(P_n^T x)))) \tag{2.13}$$

where $P_n \in \mathbb{R}^{n\times(n+l)}$ and $Q_m \in \mathbb{R}^{m\times(n+l)}$ are the matrices that project an $(n+l)$-vector onto its first n and last m components, respectively. The derivatives of the state transformations $\nabla\Phi_i$ can be explicitly calculated and written as

$$A_i := \nabla\Phi_i = I + e_{n+i}(\nabla\varphi_i(u_i) - e_{n+i})^T \in \mathbb{R}^{(n+l)\times(n+l)} \tag{2.14}$$

Now (2.13) can be differentiated using the chain rule:

$$\dot{y} = Q_m A_l A_{l-1} \ldots A_2 A_1 P_n^T \dot{x} \tag{2.15}$$

Thus, the jacobian of f can be written as

$$\frac{df}{dx}(x) = Q_m A_l A_{l-1} \ldots A_2 A_1 P_n^T \tag{2.16}$$

By transposing the product we obtain the adjoint relation

$$\bar{x} = P_n A_1^T A_2^T \ldots A_{l-1}^T A_l^T Q_m^T \bar{y} = \left(\frac{df}{dx}\right)^T \bar{y} \tag{2.17}$$

with $\bar{x} \in \mathbb{R}^n, \bar{y} \in \mathbb{R}^m$ and the identity

$$\bar{y}^T \dot{y} = \bar{x}^T \dot{x}.$$

Let us look at equation (2.17) in more detail. Suppose we have a vector $\bar{\mathbf{v}}_i$ of adjoint quantities $\bar{v}_j, 1-n \le j \le l$ such that

$$\bar{\mathbf{v}}_i = A_i^T A_{i-1}^T \ldots A_{l-1}^T A_l^T Q_m^T \bar{y} = A_i^T \bar{\mathbf{v}}_{i-1}.$$

With definition (2.14) this becomes

$$\bar{\mathbf{v}}_i = \bar{\mathbf{v}}_{i-1} + (\nabla\varphi_i(u_i) - e_{n+i})e_{n+i}^T\bar{\mathbf{v}}_{i-1}.$$

Now we can analyse what happens to the adjoint quantity $(\bar{\mathbf{v}}_i)_j = \bar{v}_j$ if we consider the i-th elemental function φ_i for $i = l, \ldots, 1$:

- Since $(\nabla\varphi_i(u_i))_j = 0$ and $e_{n+i}e_{n+i}^T\bar{\mathbf{v}}_{i-1} = 0$ for $i \neq j \not\prec i$, $\bar{v}_j$ is left unchanged if φ_i does not depend on v_j.
- If $i \neq j$ and $j \prec i$ then $(\nabla\varphi_i(u_i))_j = \frac{\partial\varphi_i}{\partial v_j}$ and $e_{n+i}^T\bar{\mathbf{v}}_{i-1} = \bar{v}_i$, thus, $\bar{v}_j$ is incremented by $\bar{v}_i\frac{\partial\varphi_i}{\partial v_j}$.
- $\bar{v}_i$ is set to zero.

With this information it is possible to rewrite the adjoint relation (2.17) as an evaluation procedure like it was done for the forward evaluation. Since the matrix-vector products in equation (2.17) are calculated for $i = l, l-1, \ldots, 1$ we have to go backward, or reverse, through the sequence of elementary function (2.2) - (2.4). Since the intermediate values v_i are needed they have to be computed first by evaluating the sequence of elemental functions. Summarizing this yields the Incremental Adjoint Recursion (ref. [GW08]):

$$\begin{aligned}
v_{i-n} &= x_i, && i = 1 \ldots n \\
v_i &= \varphi_i(v_j)_{j\prec i}, && i = 1 \ldots l \\
y_{m-i} &= v_{l-i}, && i = m-1 \ldots 0 \\
\bar{v}_{l-i} &= \bar{y}_{m-i} && i = 0 \ldots m-1 \\
\bar{v}_j &= \bar{v}_j + \bar{v}_i\frac{\partial}{\partial v_j}\varphi(u_i), j \prec i, && i = l \ldots 1 \\
\bar{x}_i &= \bar{v}_{i-n}, && i = n \ldots 1
\end{aligned}$$

As an input we need the vector $\bar{y}$. At the end we have $\bar{x}$, that is the matrix-vector product $\bar{x} = \nabla f(x)^T\bar{y}$.

Example 2

As an example we use the expression in equation (2.11). The Incremental Adjoint Recursion for this equation is the following:

$$
\begin{aligned}
v_{-1} &= x_1, \\
v_0 &= x_2, \\
v_1 &= \sin(v_{-1}), \\
v_2 &= e^{v_0}, \\
v_3 &= \sqrt{v_0}, \\
v_4 &= \frac{1}{v_{-1}}, \\
v_5 &= v_1 + v_2, \\
v_6 &= v_3 v_4, \\
y_1 &= v_5, \\
y_2 &= v_6 \\
\bar{v}_6 &+= \bar{y}_2, \\
\bar{v}_5 &+= \bar{y}_1, \\
\bar{v}_4 &+= \bar{v}_6 v_3, \\
\bar{v}_3 &+= \bar{v}_6 v_4, \\
\bar{v}_1 &+= \bar{v}_5, \\
\bar{v}_2 &+= \bar{v}_5, \\
\bar{v}_{-1} &+= -\bar{v}_4 \frac{1}{v_{-1}^2}, \\
\bar{v}_0 &+= \bar{v}_3 \frac{1}{2\sqrt{v_0}}, \\
\bar{v}_0 &+= \bar{v}_2 e^{v_0}, \\
\bar{v}_{-1} &+= \bar{v}_1 \cos(v_{-1}), \\
\bar{x}_2 &= \bar{v}_0, \\
\bar{x}_1 &= \bar{v}_{-1}.
\end{aligned}
$$

It is assumed that each $\bar{v}_i$ has been initialized with 0. Similar to the forward propagation we have to set $\bar{y} = e_i$ for $i = 1, 2$ to accumulate the full Jacobian. But here in general m calls of the evaluation trace are necessary. Thus, if we have a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and $m \ll n$, the

reverse propagation is the method of choice for the efficient calculation of the Jacobian.

2.3. Implementation and Software

The generation of the Tangent Recursion or of the Adjoint Recursion for the forward and reverse propagation, respectively, is of pure mechanical fashion provided that the derivatives of the elemental functions are known. To incorporate this fact into an actual application there exist two basic computer science concepts, namely *Source Code Transformation* and *Operator Overloading*. For the first method the source code of the function to be differentiated is augmented by derivative assignments according to section 2.1 or section 2.2. This can be done either by hand or with a preprocessor.

Since for this work the second approach is used it will be explained in a slightly more detailed way. First, consider the forward mode. In order to calculate the derivative of a function we define a new data type or class that contains the numerical value of v_i and $\dot{v}_i$. Then overloaded versions of the arithmetic operations for scalars are implemented. These overloaded operations must manipulate $\dot{v}_i$ according to the chain rule. As a last step any floating point program variable whose derivatives are needed is redeclared to be of the this new type of class. Then the gradient information of a function is generated along with the execution of this function if the derivatives of the inputs are properly initialized. Figure 2.1 shows a typical calling sequence do get the derivative of a function with respect to x_1. The function `setdotvalue` is used to set the values for $\dot{x}_1$ and $\dot{x}_2$. `getdotvalue` retrieves the derivative of the corresponding variable. In this case we would have the value $\frac{\partial y}{\partial x_1}(5, 2)$ in `dy`.

```
x_1 = 5;
x_2 = 2;
setdotvalue(x_1,1);
setdotvalue(x_2,0);
y = function(x_1,x_2);
dy = getdotvalue(y);
```

Figure 2.1.: Typical calling sequence for the forward mode with Operator Overloading

Now let us consider the reverse mode. Like for the forward mode we introduce a new data type. But now it contains the numerical value and an identifier or index. During the execution of the evaluation procedure we build up an internal representation of the computation, which we call *tape*. The tape consists of an array of operation codes and a sequence of variable indices, both encoded as integers. Furthermore there is a numerical record to store the overwritten

variables. Then, again, we define arithmetic operations of this new type that correspond to the usual floating point operations. These operations calculate the floating point operations for v_i as usual, but as a side effect they also record themselves and their arguments on the tape. We can then define a routine that reverses through the tape and calculates the adjoint variables $\bar{v}_i$ correctly according to the Incremental Adjoint Recursion from section 2.2. Just like in the forward implementation all floating point program variables must be redeclared to be of the new data type. Figure 2.2 shows the typical calling sequence for the reverse mode. First we have to specify which variable we want to tape by calling the member function `makeActive`. This basically tells the variable to record their operations on the tape. Then after calling the function we set $\bar{y}$ and can reverse through the recorded tape with the function `interpret_Adjoint`. Finally we can retrieve the gradient information from $\bar{x}_1$ and $\bar{x}_2$. In the case of the example in figure 2.2 we have $\frac{\partial y}{\partial x_1}(5,2)$ in `dx_1` and $\frac{\partial y}{\partial x_2}(5,2)$ in `dx_2`.

```
x_1 = 5;
x_2 = 2;
tape.makeActive(x_1);
tape.makeActive(x_2);
y = function(x_1,x_2);
setbarvalue(y,1);
tape.interpret_Adjoint();
dx_1 = getbarvalue(x_1);
dx_2 = getbarvalue(x_2);
```

Figure 2.2.: Typical calling sequence for the reverse mode with Operator Overloading

The forward mode by operator overloading and the reverse mode by operator overloading can be combined to obtain second order derivatives. This can be achieved by applying the General Evaluation Procedure in section 2.1 to the Incremental Adjoint Recursion of section 2.2. In terms of operator overloading this means introducing again a new variable containing values $(v_i, \bar{v}_i)$ and $(\dot{v}_i, \dot{\bar{v}}_i)$. Of course this also means that we need overloaded versions of the floating point operations as well as of the taping and interpreting functions. See figure 2.3 for a typical calling sequence. At the end we have the value $\frac{\partial^2 y}{\partial x_1^2}(5,2)$ in `ddy`.

In this work a differentiated version of the `deal.II`-C++ library (see section 3.3) is used where all the aspects mentioned above are implemented using the tool `dco` [LLN12]. Therefore it is possible to compute the gradient (or products of the gradient) of an arbitrary function created within the context of this library by following the steps mentioned above. Likewise for second order derivatives.

```
x_1 = 5;
x_2 = 2;
setdotvalue(x_1,1);
setdotvalue(x_2,0);
tape.makeActive(x_1);
tape.makeActive(x_2);
y = function(x_1,x_2);
setbarvalue(y,1);
tape.interpret_Adjoint();
dx_1 = getbarvalue(x_1);
dx_2 = getbarvalue(x_2);
ddy = getdotbarvalue(x_1);
```

Figure 2.3.: Typical calling sequence for the forward over reverse mode with Operator Overloading

Chapter 3.

Numerical Solution of the Non-linear Poisson equation

In this chapter the *Discontinous Galerkin method* will be used for the discretization of the non-linear Poisson equation to derive a discrete non-linear system. We will apply a fixed-point iteration (*Newton's method*) to get an approximate solution of that system. This rather simple approach possesses a special structure that can be exploited in the formulation of the *Discrete Adjoint method* in chapter 4. Furthermore a block-structured assembling of the Jacobian of the non-linear system is presented which utilizes *Algorithmic Differentiation* introduced in the previous chapter. The implementation is based on the `deal.II` open source library that will be briefly introduced.

3.1. Discontinous Galerkin Discretization

This section introduces the concepts to build an *Discontinous Galerkin* (DG) approximation for the non-linear homogeneous Poisson equation. Although DG methods have existed in various forms for more than 30 years, they experienced a vast development in the last decade. Originally introduced for hyperbolic problems they also became quiet popular for the solution of elliptic problems. These methods can be viewed as an extension of the standard Galerkin methods allowing for discontinuities in the ansatz space. This leads to compact disretization stencils and an substantial amount of flexibility. Figure 3.1 depicts the difference between the approaches by showing the support points of nodal basis functions for the ansatz space on two elements. For the standard Galerkin method the basis is constructed so that the basis functions are non-zero on elements that share the associated support point. For the discontinuous Galerkin method the points are logically positioned inside of each element meaning that they

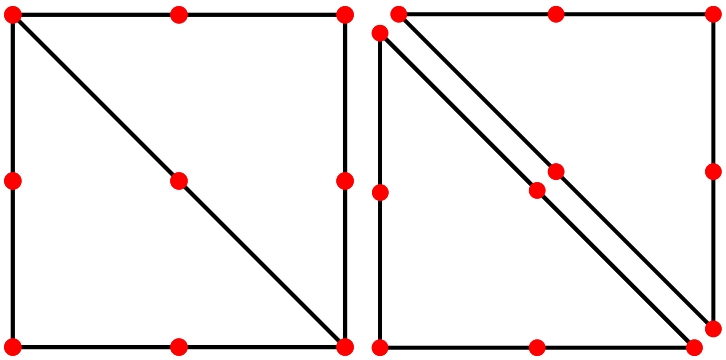


Figure 3.1.: Logical position of the support points for nodal basis functions when using the standard *Galerkin method* (left) and the *discontinuous Galerkin method* (right).

are non-zero only on one specific element. Though physically they are still on the boundary of that element. Due to this construction the problem consists of more degrees of freedom, but usually the other advantages outweigh the increased complexity.

Now consider the *Poisson* problem

$$\begin{aligned} -\Delta y + f(y) &= v, \quad \text{on } \Omega \subset \mathbb{R}^d \\ \partial_n y + k(y) &= d(u), \quad \text{on } \partial\Omega \end{aligned} \tag{3.1}$$

with a source term $v \in L^2(\Omega)$ and a boundary function $u \in L^2(\partial\Omega)$. The functionals k, d and f are assumed to be non-linear. In contrast to the standard Galerkin method for elliptic problems in the DG discretization a numerical flux is introduced, in analogy to hyperbolic problems. We will therefore derive the so called *Primal Flux Formulation* of problem (3.1) and then use the *Symmetric Interior Penalty method* for the numerical flux. This approach is for example detailed by Hartmann [Har08].

3.1.1. Discretization of the domain

The first step is to discretize the domain Ω using a mesh. We assume that the domain is Ω is an open, connected and bounded subset of $\mathbb{R}^d$ and its boundary $\Gamma := \partial\Omega$ is a finite union of parts of hyperplanes. In this case the domain can be exactly covered by a mesh consisting of polyhedral elements. We restrict ourselves to simplicial meshes $\mathcal{T}$ of the domain Ω. That is, $\mathcal{T}$ is a finite collection of disjoint non-degenerate simplices forming a partition of Ω:

$$\Omega = \bigcup_{T \in \mathcal{T}} T. \tag{3.2}$$

For later use it is necessary to uniquely identify the simplices, thus we rewrite equation (3.2) as

$$\Omega = \bigcup_{k=1}^{K} T_k, \quad T_k \in \mathcal{T}, \quad K := |\mathcal{T}|. \tag{3.3}$$

The union of all internal edges ∂T is denoted by $\Gamma_{\mathcal{I}}$, thus

$$\Gamma_{\mathcal{I}} := \bigcup_{T \in \mathcal{T}} \partial T \backslash \Gamma. \tag{3.4}$$

Furthermore let h_T be the diameter of T, the meshsize h is defined as

$$h := \max_{T \in \mathcal{T}} h_T \tag{3.5}$$

We write $\mathcal{T}_h$ to indicate the meshsize.

3.1.2. Derivation of the primal flux formulation

Let $H^m(\mathcal{T}_h)$ be the broken Sobolev space of functions $v \in L^2(\Omega)$ whose restriction on one element $T \in \mathcal{T}_h$ belongs to the sobolev space $H^m(T)$:

$$H^m(\mathcal{T}_h) := \{v \in L^2(\Omega) | \forall T \in \mathcal{T}_h, v_{|T} \in H^m(T)\} \tag{3.6}$$

The corresponding vector valued broken Sobolev space of dimension d is defined as $[H^m(\mathcal{T}_h)]^d$. To begin with the equations are transformed into a first order system of PDEs. For this purpose a function $\sigma \in [H^1(\mathcal{T}_h)]^d$ is introduced that is defined as

$$\sigma = \nabla y. \tag{3.7}$$

Inserting this into the PDE (3.1) results in

$$-\nabla \cdot \sigma + f(y) = v, \quad x \in \Omega. \tag{3.8}$$

Under the assumption $y \in H^2(\mathcal{T}_h)$ the equations (3.7) and (3.8) are multiplied with test functions $\tau \in [H^1(\mathcal{T}_h)]^d$ and $w \in H^1(\mathcal{T}_h)$, respectively, and integrated over an element $T \in \mathcal{T}_h$:

$$\int_T \sigma \cdot \tau d\vec{x} = \underbrace{\int_T \nabla y \cdot \tau d\vec{x}}_{I} \tag{3.9}$$

$$-\underbrace{\int_T \nabla \cdot \sigma w d\vec{x}}_{II} + \int_T f(y) w d\vec{x} = \int_T v w d\vec{x} \tag{3.10}$$

Integration by parts applied to the integrals I and II and reordering yields

$$\int_T \sigma \cdot \tau d\vec{x} = -\int_T y \nabla \cdot \tau d\vec{x} + \int_{\partial T} y \tau \cdot \vec{n} ds \tag{3.11}$$

$$\int_T \sigma \nabla \cdot w d\vec{x} = \int_T v w d\vec{x} - \int_T f(y) w d\vec{x} + \int_{\partial T} \sigma \cdot \vec{n} w ds, \tag{3.12}$$

where $\vec{n}$ is the outward normal unit vector on ∂T. The next step is the summation over all elements $T \in \mathcal{T}_h$. Note that the values for y and τ are not unique on $\Gamma_{\mathcal{I}}$ because the basis functions are discontinuous across elements. Therefore y and τ are replaced by numerical flux functions $\hat{y}$ and $\hat{\tau}$ on the boundaries of the elements. This results in the problem of finding $y \in H^2(\mathcal{T}_h)$ and $\sigma \in [H^1(\mathcal{T}_h)]^d$ such that

$$\int_\Omega \sigma \cdot \tau d\vec{x} = -\int_\Omega y \nabla_h \cdot \tau d\vec{x} + \sum_{T \in \mathcal{T}_h} \int_{\partial T} \hat{y} \tau \cdot \vec{n} ds, \quad \forall \tau \in [H^1(\mathcal{T}_h)]^d, \tag{3.13}$$

$$\int_\Omega \sigma \nabla_h \cdot w d\vec{x} = \int_\Omega v w d\vec{x} - \int_\Omega f(y) w d\vec{x} + \sum_{T \in \mathcal{T}_h} \int_{\partial T} \hat{\sigma} \cdot \vec{n} w ds, \quad \forall v \in H^1(\mathcal{T}_h). \tag{3.14}$$

$\hat{y}(\cdot) : H^1(\mathcal{T}_h) \to \prod_{T \in \mathcal{T}_h} L^2(\partial T)$ is a scalar numerical flux function and $\hat{\sigma}(\cdot,\cdot) : H^2(\mathcal{T}_h) \times [H^1(\mathcal{T}_h)]^d \to [\prod_{T \in \mathcal{T}_h} L^2(\partial T)]^d$ a vector valued numerical flux function. Depending on the choice of flux functions DG schemes with different properties regarding stability and accuracy arise. $\nabla_h \cdot : [H^1(\mathcal{T}_h)]^d \to L^2(\mathcal{T}_h)$ is the broken divergence operator and defined as

$$(\nabla_h \cdot \tau)|_T := \nabla \cdot (\tau|_T), \quad T \in \mathcal{T}_h \tag{3.15}$$

for $\tau \in [H^1(\mathcal{T}_h)]^d$. In a similar manner the broken gradient and Laplace operators $\nabla_h : H^1(\mathcal{T}_h) \to [L^2(\mathcal{T}_h)]^d$ and $\Delta_h : H^2(\mathcal{T}_h) \to L^2(\mathcal{T}_h)$ are defined.

To reduce the problem size of the system (3.13) - (3.14) the variable σ can be eliminated. An additional integration by parts applied to the first integral on the right hand side of equation

(3.13) and setting $\tau := \nabla_h w$ yields

$$\int_\Omega \sigma \cdot \nabla_h w d\vec{x} = \int_\Omega \nabla_h y \cdot \nabla_h w d\vec{x} + \sum_{T\in\mathcal{T}_h} \int_{\partial T} (\hat{y} - y)\vec{n} \cdot \nabla_h w ds. \tag{3.16}$$

This can be inserted into equation (3.14) which results in the problem:

Find $y \in H^2(\mathcal{T}_h)$ such that

$$\begin{aligned}\int_\Omega \nabla_h y \cdot \nabla_h w d\vec{x} - \sum_{T\in\mathcal{T}_h} \int_{\partial T} \hat{\sigma} \cdot \vec{n} w ds + \sum_{T\in\mathcal{T}_h} \int_{\partial T} (\hat{y} - y)\vec{n} \cdot \nabla_h w ds \\ = \int_\Omega v w d\vec{x} - \int_\Omega f(y) w d\vec{x}, \quad \forall w \in H^2(\mathcal{T}_h).\end{aligned} \tag{3.17}$$

This formulation is called *primal flux formulation* of the problem (3.1). The boundary terms are included in the definition of the numerical flux functions.

3.1.3. Edge based formulation

In the element based formulation (3.17) each edge $e \in \Gamma_\mathcal{I}$ is treated twice: once in $\int_{\partial T_1}$ and once in $\int_{\partial T_2}$ for $T_1 \neq T_2$ and $e = \partial T_1 \cap T_2 \neq 0$. For the implementation it is advantageous to have an edge based formulation where each edge is treated only once.

Jump and mean value operators

For a consistent depiction of a DG discretization it is reasonable to introduce some notations. On an edge $e \in \Gamma_\mathcal{I}$ between two neighbouring elements T^+ and T^- there exist two different traces q^+ and q^- for a function $q \in \prod_{T\in\mathcal{T}_h} L^2(\partial T)$ because of the possible discontinuity of the basis functions. They are defined through approaching from the directions $-\vec{n}^-$ and $-\vec{n}^+$, respectively. This is illustrated in figure 3.2. The mean value operator for functions $q \in \prod_{T\in\mathcal{T}_h} L^2(\partial T)$ and $\Phi \in [\prod_{T\in\mathcal{T}_h} L^2(\partial T)]^d$ is then defined as

$$\{\{q\}\} := \frac{1}{2}(q^+ + q^-), \quad \{\{\Phi\}\} := \frac{1}{2}(\Phi^+ + \Phi^-), \tag{3.18}$$

and the jump operator is defined as

$$[\![q]\!] := q^+\vec{n}^+ + q^-\vec{n}^-, \quad [\![\Phi]\!] := \Phi^+ \cdot \vec{n}^+ + \Phi^- \cdot \vec{n}^-. \tag{3.19}$$

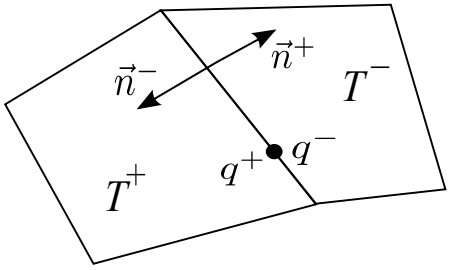


Figure 3.2.: Traces $q^+, q^- \in L^2(\partial T)$ of a function $q \in \prod_{T\in\mathcal{T}_h} L^2(\partial T)$ and the outward unit normal vectors $\vec{n}^+, \vec{n}^-$ of the elements T^+ and T^-, respectively.

For edges $e \in \Gamma$ of the domain we define $q^- \equiv 0$ and $\Phi^- \equiv 0$. Then it can be shown that

$$\sum_{T\in\mathcal{T}_h} \int_{\partial T} \Phi^+ \cdot \vec{n}^+ q^+ ds = \int_{\Gamma_\mathcal{I}\cup\Gamma} \{\{\Phi\}\} \cdot [\![q]\!] ds + \int_{\Gamma_\mathcal{I}} \{\{q\}\}[\![\Phi]\!] ds \tag{3.20}$$

holds.

Using equation (3.20) on equation (3.17) with $q = w$, $\Phi = \hat{\sigma}$ and with $q = \hat{y} - y$, $\Phi = \nabla_h w$ yields the edge based formulation of the problem (3.17):

Find $y \in H^2(\mathcal{T}_h)$ such that

$$\begin{aligned}\int_\Omega \nabla_h y \cdot \nabla_h w d\vec{x} + \int_{\Gamma_\mathcal{I}\cup\Gamma} ([\![\hat{y} - y]\!] \cdot \{\{\nabla_h w\}\} - \{\{\hat{\sigma}\}\} \cdot [\![w]\!]) ds \\ + \int_{\Gamma_\mathcal{I}} (\{\{\hat{y} - y\}\} \cdot [\![\nabla_h w]\!] - [\![\hat{\sigma}]\!]\{\{w\}\}) ds \\ = \int_\Omega v w d\vec{x} - \int_\Omega f(y) w d\vec{x}, \quad \forall w \in H^2(\mathcal{T}_h).\end{aligned} \tag{3.21}$$

3.1.4. Symmetric interior penalty method

In order to derive suitable numerical flux functions one can aim at methods that satisfy a consistency requirement and preserve discrete coercivity. This guarantees that the resulting discrete problem will be well-posed. The detailed derivation can be found in [DPE11]. Let $b : H^1(\mathcal{T}_h) \times H^1(\mathcal{T}_h) \to \mathbb{R}$ be defined as

$$\begin{aligned}b(y_h, w_h) := \int_\Omega \nabla_h y_h \cdot \nabla_h w_h d\vec{x} + \int_{\Gamma_\mathcal{I}\cup\Gamma} ([\![\hat{y} - y_h]\!] \cdot \{\{\nabla_h w_h\}\} - \{\{\hat{\sigma}\}\} \cdot [\![w_h]\!]) ds \\ + \int_{\Gamma_\mathcal{I}} (\{\{\hat{y} - y\}\} \cdot [\![\nabla_h w]\!] - [\![\hat{\sigma}]\!]\{\{w\}\}) ds\end{aligned} \tag{3.22}$$

with $y_h, w_h \in V_h \subset H^1(\mathcal{T}_h)$ and V_h being an approximation of $H^1(\mathcal{T}_h)$ such that the discrete version of problem (3.21) becomes: Find $y_h, w_h \in V_h$ such that

$$b(y_h, w_h) = \int_\Omega v w_h d\vec{x} - \int_\Omega f(y_h) w_h d\vec{x}, \quad \forall w_h \in V_h. \tag{3.23}$$

For the symmetric interior penalty (SIP) method then holds the following for the numerical fluxes (e.g. [Har08]):

$$\hat{y} = \{\{y_h\}\}, \qquad \hat{\sigma} = \{\{\nabla_h y_h\}\} - \delta^{ip}(y_h) \qquad \text{on } \Gamma_{\mathcal{I}} \tag{3.24}$$

$$\hat{y} = y_h, \qquad \hat{\sigma} = (k(y_h) - d(u))\vec{n}, \qquad \text{on } \Gamma \tag{3.25}$$

and

$$\delta^{ip}(y_h) = C_{ip}\frac{p^2}{h}[\![y_h]\!] = \delta[\![y_h]\!] \tag{3.26}$$

where C_{ip} is a problem-dependent constant, p the polynomial degree and h the diameter of the triangulation. It can easily be verified that the jump and mean-value operators have the properties

$$\{\{\{\cdot\}\}\} = \{\{\cdot\}\}, \quad \{\{[\![\cdot]\!]\}\} = [\![\cdot]\!], \quad [\![[\![\cdot]\!]]\!] = 0, \quad [\![\{\{\cdot\}\}]\!] = 0, \tag{3.27}$$

we get by inserting the definitions of the numerical fluxes into $b(y_h, w_h)$:

$$\begin{aligned} b(y_h, w_h) := \int_\Omega \nabla_h y_h \cdot \nabla_h w_h d\vec{x} + \int_{\Gamma_{\mathcal{I}}} (-[\![y_h]\!] \cdot \{\{\nabla_h w_h\}\} - \{\{\nabla_h y_h\}\} \cdot [\![w_h]\!]) ds \\ + \int_{\Gamma_{\mathcal{I}}} \delta[\![y_h]\!] \cdot [\![w_h]\!] ds + \int_\Gamma (k(y_h) - d(u)) w_h ds \end{aligned} \tag{3.28}$$

3.1.5. Vector representation

We can split $b(y_h, w_h)$ into volume, interface and face contributions as follows: For all $y_h, w_h \in V_h$,

$$b(y_h, w_h) = b^v(y_h, w_h) + b^{if}(y_h, w_h) + b^{bf}(y_h, w_h) \tag{3.29}$$

where

$$b^v(y_h, w_h) = \int_\Omega \nabla_h y_h \cdot \nabla_h w_h d\vec{x} \tag{3.30}$$

$$b^{if}(y_h, w_h) = \int_{\Gamma_{\mathcal{I}}} (-[\![y_h]\!] \cdot \{\{\nabla_h w_h\}\} - \{\{\nabla_h y_h\}\} \cdot [\![w_h]\!]) + \delta[\![y_h]\!] \cdot [\![w_h]\!] ds \tag{3.31}$$

$$b^{bf}(y_h, w_h) = \int_\Gamma (k(y_h) - d(u)) w_h ds \tag{3.32}$$

For the implementation or rather for solving the problem it is necessary to have an explicit representation for $V_h \subset H^1(\mathcal{T}_h)$. In the context of finite element methods polynomial function spaces are the most common ones, especially for DG methods the broken polynomial space consisting of piecewise continuous polynomials of degree at most p,

$$V_h := \{v \in L^2(\Omega) | \forall T \in \mathcal{T}_h, v|_T \in \mathbb{P}_d^p(T)\} \tag{3.33}$$

is used, where d denotes the dimension of the physical space. The dimension of V_h, that is, the total number of degrees of freedom N is then defined as

$$N := \dim V_h = K \times \dim \mathbb{P}_d^p(T) = K \times M. \tag{3.34}$$

By choosing an appropriate nodal basis, each function $z_h \in V_h$ can be represented as

$$z_h(x) = \sum_{i=1}^{M} z_i^k \rho_i^k(x), \quad x \in T_k, \tag{3.35}$$

where $z_i^k \in \mathbb{R}$, $\rho_i^k \in V_h$ and $V_h = \text{span}\{\rho_1^1, \rho_2^1, \ldots, \rho_M^1, \rho_1^2, \ldots, \rho_M^K\}$. Instead of testing with all $w_h \in V_h$ in the discrete problem (3.23) it is sufficient to test with all basis functions ρ_j^k. The global stiffness vector $B \in \mathbb{R}^N$ for given y_h is then of the following form:

$$B := (B_{T_1}, B_{T_2}, \ldots, B_{T_K})^T, \tag{3.36}$$

with

$$B_{T_k} := (b(y_h, \rho_1^k), b(y_h, \rho_2^k), \ldots, b(y_h, \rho_M^k)) \in \mathbb{R}^M. \tag{3.37}$$

The right hand side of problem (3.23) yields a vector $C \in \mathbb{R}^N$ similar to the structure of B but with volume contributions only, i.e.

$$C := (C_{T_1}, C_{T_2}, \ldots, C_{T_K})^T \tag{3.38}$$

with

$$C_{T_k} := (c(y_h, \rho_1^k), c(y_h, \rho_2^k), \ldots, c(y_h, \rho_M^k))^T \in \mathbb{R}^M, \tag{3.39}$$

$$c(y_h, \rho_j^k) := \int_{T_k} v \rho_j^k d\vec{x} - \int_{T_k} f \left(\sum_{i=1}^{M} y_i^k \rho_i^k \right) \rho_j^k d\vec{x}. \tag{3.40}$$

By noting that the support of ρ_j^k is just the element T_k and by using the basis representation (3.35) for y_h we can rewrite the face and boundary contributions as

$$b^v(y_h, \rho_j^k) = \sum_{i=1}^{M} \int_{T_k} y_i^k (\nabla_h \rho_i^k \cdot \nabla_h \rho_j^k) d\vec{x}, \quad j = 1, \dots, M \tag{3.41}$$

$$b^{bf}(y_h, \rho_j^k) = \int_{e \in \partial T_k \cap \Gamma} \left[k \left(\sum_{i=1}^{M} y_i^k \rho_i^k \right) - d(u) \right] \rho_j^k ds, \quad j = 1, \dots, M \tag{3.42}$$

A bit more complicated is the interface contribution, since an edge e contributes to two blocks $B_{T_{k^+}}$ and $B_{T_{k^-}}$, where $k^+ := k$ and k^- are the indices of the elements which share the edge e. By inserting the definitions of the jump and mean-value operators we get for each edge $e \in \Gamma_{\mathcal{I}}$:

$$\begin{aligned}
&\int_e \sum_{i=1}^{M} (-[\![y_i \rho_i]\!] \cdot \{\{\nabla_h \rho_j\}\} - \{\{y_i \nabla_h \rho_i\}\} \cdot [\![\rho_j]\!] + \delta [\![y_i \rho_i]\!] \cdot [\![\rho_j]\!]) ds \\
&= \int_e \sum_{i=1}^{M} (-y_i^+ \rho_i^+ \vec{n}^+ \cdot \nabla_h \rho_j^+ - y_i^- \rho_i^- \vec{n}^- \cdot \nabla_h \rho_j^+ - y_i^+ \rho_i^+ \vec{n}^+ \cdot \nabla_h \rho_j^- - y_i^- \rho_i^- \vec{n}^- \cdot \nabla_h \rho_j^- \\
&\quad - \rho_j^+ \vec{n}^+ \cdot y_i^+ \nabla_h \rho_i^+ - \rho_j^- \vec{n}^- \cdot y_i^+ \nabla_h \rho_i^+ - \rho_j^+ \vec{n}^+ \cdot y_i^- \nabla_h \rho_i^- - \rho_j^- \vec{n}^- \cdot y_i^- \nabla_h \rho_i^- \\
&\quad + 2\delta [(y_i^+ \rho_i^+ \vec{n}^+ \cdot \rho_j^+ \vec{n}^+) + (y_i^+ \rho_i^+ \vec{n}^+ \cdot \rho_j^- \vec{n}^-) + (y_i^- \rho_i^- \vec{n}^- \cdot \rho_j^+ \vec{n}^+) + (y_i^- \rho_i^- \vec{n}^- \cdot \rho_j^- \vec{n}^-)]) ds
\end{aligned}$$

where the indices $+$ and $-$ mean k^+ and k^-, respectively. Furthermore it is advantageous to introduce the following abbreviations:

$$\begin{aligned}
\rho_{ij}^{++} &:= -\rho_i^+ \vec{n}^+ \cdot \nabla_h \rho_j^+ - \rho_j^+ \vec{n}^+ \cdot \nabla_h \rho_i^+ + 2\delta(\rho_i^+ \vec{n}^+ \cdot \rho_j^+ \vec{n}^+) \\
\rho_{ij}^{+-} &:= -\rho_i^+ \vec{n}^+ \cdot \nabla_h \rho_j^- - \rho_j^- \vec{n}^- \cdot \nabla_h \rho_i^+ + 2\delta(\rho_i^+ \vec{n}^+ \cdot \rho_j^- \vec{n}^-) \\
\rho_{ij}^{-+} &:= -\rho_i^- \vec{n}^- \cdot \nabla_h \rho_j^+ - \rho_j^+ \vec{n}^+ \cdot \nabla_h \rho_i^- + 2\delta(\rho_i^- \vec{n}^- \cdot \rho_j^+ \vec{n}^+) \\
\rho_{ij}^{--} &:= -\rho_i^- \vec{n}^- \cdot \nabla_h \rho_j^- - \rho_j^- \vec{n}^- \cdot \nabla_h \rho_i^- + 2\delta(\rho_i^- \vec{n}^- \cdot \rho_j^- \vec{n}^-)
\end{aligned}$$

Hence for each edge $e \in \Gamma_{\mathcal{I}}$ we get two local contributions, namely

$$b^{if}(y_h, \rho_j^{k^+}) = \sum_{i=1}^{M} \int_e y_i^+ \rho_{ij}^{++} + y_i^- \rho_{ij}^{-+} ds, \quad j = 1, \dots, M \tag{3.43}$$

$$b^{if}(y_h, \rho_j^{k^-}) = \sum_{i=1}^{M} \int_e y_i^- \rho_{ij}^{--} + y_i^+ \rho_{ij}^{+-} ds, \quad j = 1, \dots, M. \tag{3.44}$$

For given y_h we can simply loop over all $T_k \in \mathcal{T}$ and add the face and boundary contributions (3.41) and (3.42), respectively, to the corresponding entries in B. The same can be done for the vector C. For the interface contributions we need a separate loop over all edges $e \in \Gamma_{\mathcal{I}}$

and an adding of (3.43) and (3.44) to B.

We get the vector representation of problem (3.23) if we consider C and B as functions that depend on the vector $\tilde{y} := (y_1^1, y_2^1, \ldots, y_j^k, \ldots, y_M^K)^T \in \mathbb{R}^N$. Then we want to find $\tilde{y}$ such that

$$B(\tilde{y}) - C(\tilde{y}) = 0. \tag{3.45}$$

Up to now we considered the boundary function u and the source function v as determined by the problem. But for the optimal control problem in section 4.1 we need approximations u_h, v_h. For v_h we can simply use the same ansatz space V_h we used for y_h. Therefore we can represent $v_h \in V_h$ as

$$v_h(x) = \sum_{i=1}^{M} v_i^k \rho_i^k(x), \quad x \in T_k. \tag{3.46}$$

For u_h we define the space U_h with

$$U_h := \{u \in \partial L^2(\Omega) | \forall e \in \Gamma, u|_e \in \mathbb{P}_d^p(e)\} \tag{3.47}$$

such that

$$u_h(x) = \sum_{i=1}^{L} u_i^k \psi_i^k(x), \quad x \in \Gamma \cap \partial T_k. \tag{3.48}$$

with $U_h = \text{span}\{\psi_1^1, \psi_2^1, \ldots, \psi_L^1, \psi_1^2, \ldots, \psi_L^K\}$. If we use this approximations for u and v in equation (3.45) and by defining the vector

$$\tilde{u} := (u_1^1, u_2^1, \ldots, u_L^1, u_1^2, \ldots, u_L^K, v_1^1, v_2^1, \ldots, v_M^1, v_1^2, \ldots, v_M^K)^T \in \mathbb{R}^{K(M+L)} \tag{3.49}$$

we get the following problem: Find $\tilde{y} \in \mathbb{R}^N$ for given $\tilde{u} \in \mathbb{R}^{K(M+L)}$ such that

$$B(\tilde{y}, \tilde{u}) - C(\tilde{y}, \tilde{u}) = 0. \tag{3.50}$$

It can be shown that this problem is stable and has a unique solution under some conditions on the functional f [BG05]. We assume that these conditions are satisfied in the following.

3.2. Solving the nonlinear system

3.2.1. Newton's Method

As we have seen in the last section, the discretization of the PDE yields a nonlinear system of equations that can be written as

$$F(\tilde{y}) := B(\tilde{y}, \tilde{u}) - C(\tilde{y}, \tilde{u}) = 0 \tag{3.51}$$

with $F : \mathbb{R}^N \to \mathbb{R}^N$, that implicitly defines $\tilde{y} \in \mathbb{R}^N$ for given $\tilde{u} \in \mathbb{R}^{K(M+L)}$. By using a Newton-type method we can define an iterative procedure as

$$\tilde{y}^{m+1} := \tilde{y}^m - P(\tilde{y}^m, \tilde{u})F(\tilde{y}^m, \tilde{u}), \quad m = 1, 2, \ldots, \tag{3.52}$$

were $P : \mathbb{R}^N \times \mathbb{R}^U \to \mathbb{R}^{N \times N}$ is a suitable preconditioner that approximates the inverse of the Jacobian F_y^{-1}. Now we will restrict ourselves to the standard Newton method where

$$P(\tilde{y}^m, \tilde{u}) := F_y^{-1}(\tilde{y}^m, \tilde{u}). \tag{3.53}$$

We can consider the method as converged if $\|F(\tilde{y}^m, \tilde{u})\|$ is sufficiently small. By our assumption that the system (3.51) is well-posed, we can further assume that the Jacobian F_y exists near $\tilde{y}$ and that it is invertible. Then $G(\tilde{y}, \tilde{u})$, defined as

$$G_y(\tilde{y}, \tilde{u}) := I - F_y^{-1}(\tilde{y}, \tilde{u}), \tag{3.54}$$

satisfies for all arguments $(\tilde{y}^m, \tilde{u})$ in some neighborhood of $(\tilde{y}, \tilde{u})$

$$\|G_y\| \leq \rho < 1 \tag{3.55}$$

where ρ is the maximal modulus of any eigenvalue of G_y and with respect to some induced norm $\|\cdot\|$. For a proof see for example [OR70]. If F_y is also Lipschitz continuous, recurrence (3.52) converges locally quadratic [DR08]. In each step of the Newton method (3.52) we have to multiply the inverse of the Jacobian, F_y^{-1}, with F. Since this is in general a very ill-conditioned task, we rather seek for a solution $d \in \mathbb{R}^N$ of

$$F_y(\tilde{y}^m, \tilde{u})d = F(\tilde{y}^m, \tilde{u}) \tag{3.56}$$

and then update $\tilde{y}^m$:

$$\tilde{y}^{m+1} := \tilde{y}^m - d. \tag{3.57}$$

Although we used the SIP method for the disretization the matrix F_y is non-symmetric due to the non-linear terms involving y_h. Therefore we use the *Generalized Minimal Residual (Gmres)* [SS86] method for the solution of the linear system (3.56).

3.2.2. Accumulation of the Jacobian

In order to apply recurrence (3.52) to solve the nonlinear system we need an explicit representation of the Jacobian F_y, i.e. the derivatives of the vectors B and C with respect to the vector $\tilde{y}$. A first naive approach would be the application of algorithmic differentiation to the underlying function in the program that accumulates these vectors. As was shown in section 2.1 we would need N calls of the function that assembles F to generate F_y if we use the forward mode. To clarify that there is a more efficient way, consider the derivative of $B(\tilde{y}, \tilde{u})$,

$$B_y := \begin{pmatrix} \frac{\partial B_{T_1}}{\partial \tilde{y}} \\ \frac{\partial B_{T_2}}{\partial \tilde{y}} \\ \vdots \\ \frac{\partial B_{T_K}}{\partial \tilde{y}} \end{pmatrix} \in \mathbb{R}^{N \times N} \tag{3.58}$$

with

$$\frac{\partial B_{T_k}}{\partial \tilde{y}} \in \mathbb{R}^{M \times N}. \tag{3.59}$$

Remember that each entry $(B_{T_k})_j$ is defined as

$$(B_{T_k})_j := b(y_h, \rho_j^k) \tag{3.60}$$

with $b(y_h, w_h)$ consisting of volume, boundary and interface contributions (ref. (3.29)). From equation (3.41) and (3.42) we notice that the volume and boundary contributions for ρ_j^k only depend on those y_i^m in $\tilde{y}$ for which $m = k$ holds. Furthermore due to equation (3.43) and (3.44) the interface contributions for ρ_j^k depend on those y_i^m for which $m = k$ holds and T_k and T_m share a common edge. Thus most entries in $\frac{\partial B_{T_k}}{\partial \tilde{y}}$ and therefore in B_y will be zero. The same holds for C_y, though there are no interface contributions. We then have three types of blocks, namely

$$\frac{\partial B_{T_k}}{\partial y^k} \in \mathbb{R}^{M \times M} \tag{3.61}$$

$$\frac{\partial B_{T_k}}{\partial y^m} \in \mathbb{R}^{M \times M}, \quad T_k \cap T_m \neq \emptyset, m \neq k \tag{3.62}$$

$$\frac{\partial C_{T_k}}{\partial y^k} \in \mathbb{R}^{M \times M} \tag{3.63}$$

where $y^k = (y_1^k, y_2^k, \ldots, y_M^k)^T \in \mathbb{R}^M$. Thus by applying the forward mode to the functions that assemble B_{T_k} and C_{T_k} we can calculate the blocks (3.61) - (3.63) and then construct

$$F_y(\tilde{y}, \tilde{u}) = B_y(\tilde{y}, \tilde{u}) - C_y(\tilde{y}, \tilde{u}) \tag{3.64}$$

by accumulating them, i.e. each entry of the blocks has to be added to the corresponding entry in the global Jacobian.

Now assume that the cost for one call of each function that generates B_{T_k} or C_{T_k} is $\mathcal{O}(M)$ operations. The assembling of F then needs $\mathcal{O}(KM)$ operations. By applying the forward mode directly to the function for F we would need N calls of that function and therefore $\mathcal{O}(NKM)$ operations to accumulate F_y. For the aforementioned block-wise accumulation we first assemble the element Jacobians (3.61) - (3.63) by applying the forward mode to the functions for B_{T_k} and C_{T_k}. This results in $\mathcal{O}(M^2)$ operations for each element T_k and therefore in a total of $\mathcal{O}(KM^2)$ operations for F_y. Since $NKM = K^2M^2 \ll KM^2$ the block-wise approach offers a huge improvement in terms of complexity.

3.3. `deal.II` - C++ Library

As already mentioned the `deal.II`-C++ library [BHK07] is used for the implementation of the numerical solver introduced throughout this chapter. This library is targeted at the computational solution of partial differential equations using finite elements. The program library takes care of the details of grid handling, handling of degrees of freedom, input of meshes and output of results in graphics formats, and the like. Furthermore the library comprises many different solver for linear systems including several preconditioner. Especially the aspects that were not mentioned in section 3.1.5 or 3.2.1 that are necessary for the implementation, e.g. performing numerical integration of the basis functions, are carried out by library functions. Some important features that are used in this work and worth mentioning are the following:

- The discontinuous finite elements are based on tensor products of Lagrangian polynomials. The shape functions $\rho_i^k(x)$ are Lagrangian interpolants of an equidistant grid of points on the unit cell.
- Integration is performed using the Gauss-Legendre quadrature rule of sufficiently high order.
- The mesh consists only of hypercubes.

- All linear systems involving the Jacobian are solved using the *GMRES* (generalized minimal residual) method with block preconditioning.

Chapter 4.

Optimal Control of Partial Differential Equations

The theory of optimal control deals with the problem of finding an "optimal" solution of systems whose dynamics are described by mathematical models. Tröltzsch [Trö10] considers the following ingredients as essential features of an optimal control problem: A *cost functional* that has to be minimized and which defines the optimal solution. A *state* y which is determined by the mathematical model, and a *control function* u used to alter the state. Furthermore various constraints for the state and control are possible. In fields like robotics or control of chemical processes the systems are usually modelled by ordinary differential equations. But often it is necessary to deploy partial differential equations to model physical phenomena like diffusion, electromagnetic waves, heat conduction or fluid flows. For example in the latter case there exist many interesting cost functionals. Consider the flow around an airfoil. In this case the flow can be modelled by means of the Navier-Stokes equations. In order to save fuel it is desirable to reduce the drag while the volume (or the area in 2D) of the wing should be constant to maintain the amount of fuel that can be carried. The state y is described by the flow variables, that is, the velocity and the pressure. The drag coefficient, which is the *cost functional* we want to minimize, depends on the flow variables and on the shape of the airfoil. Thus the *control* u consists of the values that describe the geometry, e.g. spline control points.

As a first rigorous example we formulate a simple problem: Consider a bounded Lipschitz domain $\Omega \subset \mathbb{R}^2$. Assume that we are given a target (or desired state) y_d which is a real valued function of Ω. Our aim is to find a function (or control) u (also defined on Ω) such that the

solution (or state) y of

$$\begin{aligned} \Delta y &= u, \text{ on } \Omega, \\ y &= 0, \text{ on } \partial\Omega. \end{aligned}$$

matches y_d at the best. Moreover our solution (or control) shall obey some pointwise bounds

$$a \leq u \leq b.$$

This motivates the following constrained optimization problem

$$\begin{aligned} &\min_{y,u} \|y - y_d\|_{L^2}^2 \\ s.t. \quad &\Delta y = u, \text{ on } \Omega, \\ &y = 0, \text{ on } \partial\Omega \\ &a \leq u \leq b. \end{aligned} \tag{4.1}$$

As can be seen from this small example the field of optimal control is closely related to the broad area of numerical optimization. The main difference is that in optimal control we have two distinct types of variables, the control u and the state y. In some sense they are treated separately, as we will point out. In general there exist many possible methods for the solution. The efficient ones are the gradient based *adjoint* methods. This term comprises two approaches for the solution. The first one is called *continuous adjoint* method while the second one is called *discrete adjoint* method. For the former one the optimality conditions are formulated in the continuous setting, which results in the so called *continuous adjoint equation*. This equation, which is an additional PDE, has then to be solved to get the corresponding *adjoint state*. For the second approach the PDE describing the state is discretized and then the optimality conditions are formulated.

4.1. Discrete Adjoint Method

We will now derive the optimality conditions for the discrete adjoint method. First, consider the general continuous optimal control problem

$$\begin{aligned} &\min_{y,u} J(y,u), \\ s.t. \quad &S(y,u) = 0, \\ &a \leq u \leq b. \end{aligned} \tag{4.2}$$

The control u may not be differentiable so a natural choice would be $u \in L^2(\Omega)$ if u is a distributed control, that is, a source term in the PDE, or $u \in L^2(\partial\Omega)$ if it is a function in the boundary conditions. In the latter case u is called boundary control. Also a combination of both control types would be possible. Since S is the differential operator of a partial differential equation for y, we assume $y \in H^2(\Omega)$. Furthermore $a, b \in \mathbb{R}$. For the discretization of the PDE $S(y, u) = 0$ we use the Discontinous Galerkin approach introduced in the previous chapter. Thus, we have the approximations

$$y(x) \approx y_h(x) = \sum_{i=1}^{N} y_i \rho_i(x), \quad u(x) \approx u_h(x) = \sum_{i=1}^{U} u_i \psi_i(x) \tag{4.3}$$

and

$$\tilde{y} = (y_1, y_2, \ldots, y_N)^T \in \mathbb{R}^N$$
$$\tilde{u} = (u_1, u_2, \ldots, u_K)^T \in \mathbb{R}^U.$$

Then we can write the discrete version of (4.2) as

$$\begin{aligned} & \min_{\tilde{y},\tilde{u}} \tilde{J}(\tilde{y}, \tilde{u}), \\ s.t. \quad & G(\tilde{y}, \tilde{u}) = \tilde{y}, \\ & a \leq \tilde{u} \leq b. \end{aligned} \tag{4.4}$$

Here $\tilde{J} : \mathbb{R}^N \times \mathbb{R}^U \to \mathbb{R}$ is the discrete version of the functional J in the continuous problem (4.2). The discretization of the PDE yields a nonlinear system of equations which is solved by applying a fixed point iteration. Since we cannot assume that this system will be solved exactly we use the fixed point statement $G(\tilde{y}, \tilde{u}) = \tilde{y}$ with $G : \mathbb{R}^N \times \mathbb{R}^U \to \mathbb{R}^N$. For example for a Newton-type method G was defined as

$$G(\tilde{y}, \tilde{u}) := \tilde{y} - P(\tilde{y}, \tilde{u}) F(\tilde{y}, \tilde{u}) \tag{4.5}$$

where $F : \mathbb{R}^N \times \mathbb{R}^U \to \mathbb{R}^N$ is the nonlinear function coming from the discretization of the PDE (see section 3.1) and $P : \mathbb{R}^N \times \mathbb{R}^U \to \mathbb{R}^N$ is a suitable preconditioner. Thus we see that the transition from the continuous problem (4.2) to the discrete problem (4.4) can be achieved by simply replacing the continuous functions and functionals by their discretized versions. For the sake of simplicity the tilde will be omitted in the following.

For the formulation of the optimality conditions we use the Lagrange approach. Therefore the

Lagrangian $L : \mathbb{R}^N \times \mathbb{R}^U \times \mathbb{R}^N \to \mathbb{R}$ with

$$L(y, u, \lambda) := J(y, u) + \lambda^T (G(y, u) - y) = N(y, u, \lambda) - \lambda^T y \tag{4.6}$$

is introduced, where $\lambda \in \mathbb{R}^N$ is the *Lagrange multiplicator* or the *adjoint* variable, and

$$N(y, u, \lambda) := J(y, u) + \lambda^T G(y, u) \tag{4.7}$$

is called the *shifted Lagrangian*. Each argument of $L(y, u, \lambda)$ is considered to be independent of the others; this was not true for the original optimization problem (4.4) involving $J(y, u)$ since there the arguments y and u were constrained to satisfy $G(y, u) = y$ and thus could not be chosen independently. The first order optimality conditions (*Karush-Kuhn-Tucker (KKT) conditions*, ref. [NW06]) can then be stated as

$$\frac{\partial L}{\partial \lambda}(y, u, \lambda) = G(y, u) - y = 0 \tag{4.8}$$

$$\frac{\partial L}{\partial y}(y, u, \lambda) = \frac{\partial N}{\partial y}(y, u, \lambda) - \lambda^T = 0 \tag{4.9}$$

$$\frac{\partial L}{\partial u}(y, u, \lambda) = \frac{\partial N}{\partial u}(y, u, \lambda) = 0 \tag{4.10}$$

or, equivalently, a KKT-point (y^*, u^*, λ^*), that is, a point where the gradient of the Lagrangian (4.6) vanishes must satisfy

$$G(y^*, u^*) = y^* \qquad \Rightarrow \text{state equation} \tag{4.11}$$

$$N_y^T(y^*, u^*, \lambda^*) = \lambda^* \qquad \Rightarrow \text{adjoint equation} \tag{4.12}$$

$$N_u(y^*, u^*, \lambda^*) = 0, \qquad \Rightarrow \text{optimality condition} \tag{4.13}$$

where the subscript denotes the derivative with respect to this variable.

The conditions for a KKT-point immediately give us an idea for a solution approach. Let us assume for a moment that we already have a y and λ that satisfy the state equation (4.11) and the adjoint equation (4.12) for a given $u \neq u^*$. Then equation (4.10) can be used to calculate the gradient of the Lagrangian L_u by calculating the gradient of the shifted Lagrangian. With this information any gradient based method for unconstrained optimization like quasi-Newton methods or Line-Search methods can be used to determine an update of the control that hopefully gives a substantial reduction in the Lagrangian L. The control box-constraints $a \leq u \leq b$ can be incorporated into this step. We can summarize this as follows:

Choose y_0, u_0, λ_0, set $k = 0$ then do

1. Calculate y_{k+1},λ_{k+1} by solving the state equation (4.11) and adjoint equation (4.12)

using u_k.

2. Compute the gradient of the shifted Lagrangian N_u.

3. Use a gradient based optimization method for the determination of an update δu_k and set $u_{k+1} := u_k + \delta u_k$.

4. If "satisfactory convergence" is achieved, then stop. Otherwise set $k := k + 1$ and go to step 1.

This general approach for the solution of the discrete optimal control problem (4.4) is shown in figure 4.1. The solution of the state equation in step 1 can be done with any Newton-type method, here we use the standard Newton method as introduced in 3.2.1. The adjoint equation can be solved with several methods. In particular in section 4.2 four different approaches are considered. The problem of finding an update δu_k in step 3 is adressed in section 4.3 where the projected gradient method will be introduced. The functions J, N and G are implemented as routines on top of the program for solving the PDE, thus all necessary derivatives are calculated via Algorithmic Differentiation by using the methods described in section 2.3.

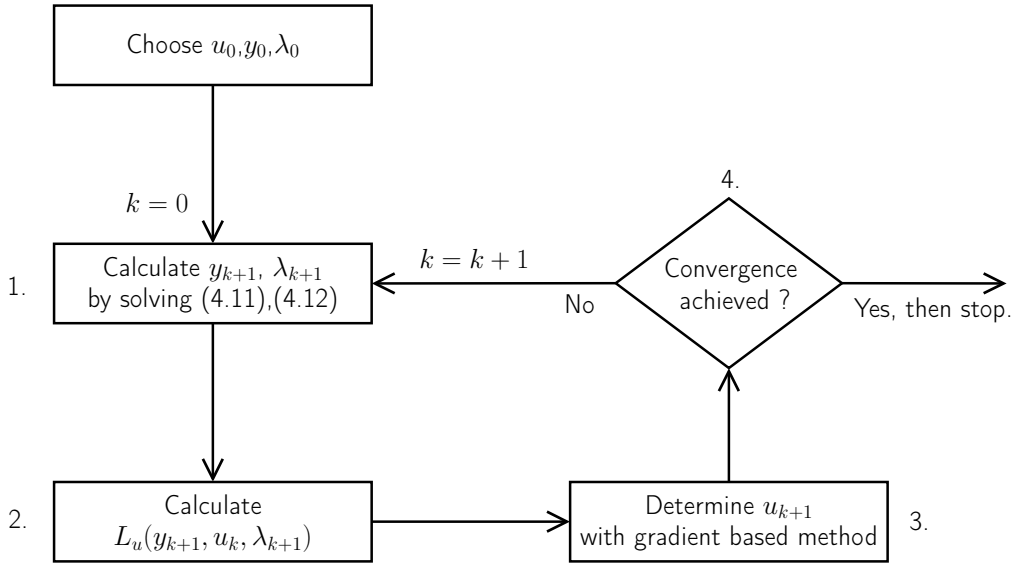


Figure 4.1.: General procedure for the solution of the optimal control problem (4.4).

4.2. Solving the adjoint equation

4.2.1. Reverse Accumulation

We use the structure of the shifted Lagrangian to derive an iterative method for the adjoint variable λ. The adjoint equation

$$N_y(y^*, u, \lambda^*) = J_y(y^*, u) + G_y^T(y^*, u)\lambda = \lambda. \tag{4.14}$$

can be rewritten as

$$(G_y^T(y^*, u) - I)\lambda^* = -J_y(y^*, u). \tag{4.15}$$

Equation (4.15) above is a linear system for λ^* if y^* is the (numerical) solution of the state equation and u is fixed. In principle any solver that can handle non-symmetric matrices would be suitable for solving this system. Another way of obtaining a solution relies upon the fact that (4.15) implies a splitting of the system matrix $G_y^T - I$ such that we obtain the iterative method

$$\lambda^{m+1} := G_y^T(y^*, u)\lambda^m + J_y(y^*, u) = N_y(y^*, u, \lambda^m), \quad m = 0, 1, \ldots \tag{4.16}$$

This recurrence converges for arbitrary λ^0 in the image of N_y if $\rho(G_y^T) < 1$. In practice that restriction does not pose a problem since it is fulfilled if the state equation (4.11) has been solved with a converging Newton-type method because then

$$\rho(G_y^T) = \rho(G_y) < 1 \tag{4.17}$$

holds (see section (3.2.1) for the last inequality). This method, known as *Reverse Accumulation*, was first analyzed in the work of Christiansion [Chr98]. There he also showed that iteration (4.16) maintains the quadratic convergence behaviour if the state equation is solved with the standard Newton approach. If this method has converged we obviously have $\lambda^m = \lambda^{m+1} = \lambda^*$, or, in other words, the residual $\|\lambda^{m+1} - \lambda^m\|$ has vanished. Therefore it seems reasonable to stop iteration (4.16) if $\|\lambda^{m+1} - \lambda^m\|$ is sufficiently small.

4.2.2. Piggy-back method

It can be shown that iteration (4.16) converges even if we replace the exact solution of the state equation y^* by an intermediate value y^m. Consequently it is possible to solve the state

and adjoint equation simultaneously. This results in the iteration

$$\left.\begin{aligned} y^{m+1} &:= G(y^m, u) \\ \lambda^{m+1} &:= N_y(y^m, u, \lambda^m) \end{aligned}\right\} \quad m = 0, 1, \ldots \tag{4.18}$$

for given $y^0, \lambda^0 \in \mathbb{R}^N$ and $u \in \mathbb{R}^U$. We can stop the iteration if both, $\|F_y(y^m, u)\|$ and $\|\lambda^{m+1} - \lambda^m\|$, are below some threshold. One practical problem is that typically the λ^m lacks a bit behind the y^m in terms of convergence. Then in order to satisfy both stopping criteria we would actually do more iterations for y^m as would be necessary to reach a certain accuracy. One has to keep that in mind as we will later compare the methods for given accuracies.

4.2.3. Two-Phase

The *Two-Phase method* or also called *black-box differentiation* deploys the principles of automatic differentiation to the optimal control problem. Suppose we iterate over $\bar{m}$ steps

$$y^{m+1} = G(y^m, u), \quad m = 0, 1, \ldots, \bar{m} - 1 \tag{4.19}$$

and then evaluate

$$z := J(y^{\bar{m}}, u). \tag{4.20}$$

with $z \in \mathbb{R}$. If we consider each iteration cycle in equation (4.19) and the evaluation of J as elemental equations we can apply the Incremental Adjoint Recursion from section 2.2 to get the adjoint $\bar{y}$:

$$\bar{y}^{\bar{m}} = J_y^T(y^{\bar{m}}, u)\bar{z} \tag{4.21}$$

and

$$\bar{y}^m = G_y^T(y^m, u)\bar{y}^{m+1} \quad m = \bar{m} - 1, \ldots, 0. \tag{4.22}$$

Note, that it is possible to represent $\bar{y}^m$ as

$$\bar{y}^m = \prod_{j=\bar{m}}^{m} G_y^T(y^j, u)\bar{y}^{\bar{m}}. \tag{4.23}$$

Since G_y is contractive, i.e. $\|G_y\| \leq \rho < 1$, where ρ is the maximal modulus of any eigenvalue of G_y, we find that

$$\|\bar{y}^m\| \sim \rho^{\bar{m}-m}. \tag{4.24}$$

Thus the adjoints $\bar{y}^m$ of early iterates should be small when $\bar{m}$ is sufficiently large. This makes sense since the initial guess y^0 should not have any significant influence on the outcome of the computation. Let us assume for a moment that $y^m = y^*$ for all $m \leq \bar{m}$ (including negative indices) and consequently $G_y(y^m, u) = G_y(y^*, u)$ for all $m \leq \bar{m}$. Then it follows from equation (4.23)

$$\bar{y}^m = [G_y^T(y^*, u)]^{\bar{m}-m}\bar{y}^{\bar{m}} = [G_y^T(y^*, u)]^{\bar{m}-m}J_y^T(y^*)\bar{z} \tag{4.25}$$

and taking the sum over all $m \leq \bar{m}$:

$$\sum_{i=0}^{\infty}\bar{y}^{\bar{m}-i} = \sum_{i=0}^{\infty}[G_y^T(y^*, u)]^i J_y^T(y^*)\bar{z} \tag{4.26}$$

The sum on the right hand side is a so called Neumann series. It can be easily be shown that

$$\sum_{i=0}^{\infty}[G_y^T(y^*, u)]^i = (I - G_y^T(y^*, u))^{-1}, \tag{4.27}$$

if $\|G_y\| < 1$ holds. Thus it follows from equation (4.26)

$$(G_y^T(y^*, u) - I)\sum_{i=0}^{\infty}\bar{y}^{\bar{m}-i} = -J_y^T(y^*)\bar{z}. \tag{4.28}$$

This is exactly the adjoint equation (4.15) if

$$\lambda^* = \sum_{i=0}^{\infty}\bar{y}^{\bar{m}-i}. \tag{4.29}$$

and $\bar{z} \equiv 1$. Without the unrealistic assumption $y^m = y^*$ we can then use a finite version of the series (4.29) as an approximation for the exact Lagrange multiplier. Hence we define

$$\lambda := \sum_{i=0}^{\bar{m}}\bar{y}^{\bar{m}-i} = \sum_{i=0}^{\bar{m}}\left[\prod_{j=\bar{m}}^{\bar{m}-i} G_y^T(y^j, u)\right]\bar{y}^{\bar{m}}. \tag{4.30}$$

Per definition equation (4.30) is then in some sense consistent with equation (4.29). In contrast to the other methods it is not possible to check the quality of the approximation by evaluating some residual that must vanish if it is correct. But since

$$\|\bar{y}^m\| \underbrace{=}_{(4.22)} \|G_y^T(y^m, u)\bar{y}^{m+1}\| \leq \|G_y^T(y^m, u)\|\|\bar{y}^{m+1}\| \leq \rho\|\bar{y}^{m+1}\| \tag{4.31}$$

declines monotonically we can expect that λ has reached its proper value if $\|\bar{y}^m\|$ become sufficiently small. Due to equation (4.24) this is the case if $\bar{m}$ is moderately large. This also

means that for small m (i.e. large i), where $G_y(y^{\bar{m}-i}, u)$ is quite far away from $G_y(y^*, u)$, the product in equation (4.30) will also be small.

4.2.4. Giles Exact Dual

The method we here call *Giles Exact Dual* has not been developed by M. B. Giles, but became popular especially for optimal design problems in computational fluid dynamics because of his article [GP00] he published with N.A. Pierce. We derive this method as a simplification of the *Reverse Accumulation* in section 4.2.1.

For the solution of the adjoint equation (4.12) we need the derivative of G with respect to the state y. If we use a Newton-type method we get with equation (4.5) and using the chain rule

$$G_y(y, u) = I - P(y, u)F_y(y, u) - P_y(y, u)F(y, u). \tag{4.32}$$

This equation is true for all $y \in \mathbb{R}^N$. If one uses $y = y^*$, i.e. the exact solution of the fixed point iteration in the discrete problem (3.23), we have $F(y^*, u) = 0$ in equation (4.5) and therefore equation (4.32) reduces to

$$G_y(y^*, u) = I - P(y^*, u)F_y(y^*, u). \tag{4.33}$$

If we use equation (4.33) for the derivative of G_y in iteration (4.16) we get the recurrence

$$\lambda^{m+1} := J_y^T(y^*, u) + (I - F_y^T(y^*, u)P^T(y^*, u))\lambda^m, \quad m = 0, 1, \ldots, \tag{4.34}$$

Similar to the derivative G_y at y^* we get for the derivative G_u needed for the gradient calculation in equation (4.13):

$$G_u(y^*, u) = -P(y^*, u)F_u(y^*, u) \tag{4.35}$$

Then for the gradient of the shifted Lagrangian holds

$$N_u^T(y^*, u, \lambda^{\bar{m}}) = J_u^T(y^*, u) - F_u^T(y^*, u)P^T(y^*, u)\lambda^* = J_u^T(y^*, u) - F_u^T(y^*, u)\tilde{\lambda}^* \tag{4.36}$$

where λ^* is the solution of the recurrence (4.34) and

$$\tilde{\lambda}^m := P^T(y^*, u)\lambda^m. \tag{4.37}$$

If we multiply recurrence (4.34) by $P^T(y^*, u)$ and by using equation (4.37) we get a recurrence for $\tilde{\lambda}^m$:

$$\tilde{\lambda}^{m+1} := \tilde{\lambda}^m + P^T(y^*, u)(J_y^T(y^*, u) - F_y^T(y^*, u)\tilde{\lambda}^m), \quad m = 0, 1, \ldots, \tag{4.38}$$

Thus by making the assumption that the non-linear system is solved exactly we completely avoided the differentiation of the preconditioner P. Note that if the standard Newton method is used, that is, if

$$P^T(y,u) := F_y^{-1}(y,u), \tag{4.39}$$

we would get second derivatives of F in equation (4.32). But in our case recurrence (4.38) then simplifies to a linear system for $\tilde{\lambda}^*$:

$$F_y(y^*,u)\tilde{\lambda}^* = J_y^T(y^*,u) \tag{4.40}$$

Up to now all equations are exact if $y = y^*$. But in practice we stop our fixed point method for y after a finite number of iterations, yielding a y^{k+1} in step 2 in figure 4.1 that may not be close to y^*. Then solving for $\tilde{\lambda}^{k+1}$ by using equation (4.40) or recurrence (4.38) with y^{k+1} might introduce a relatively larger error in the gradient that is calculated with equation (4.36). Because in general we cannot assure that $P_y(y,u)F(y,u)$ in equation (4.32) will be small. Nevertheless this method is used very often in practice because of its simplicity. Note that we would also get equation (4.40) as the adjoint equation if we would have used $F(y,u) = 0$ instead of $G(y,u) = y$ in the discrete problem (4.4) and then formulated the optimality conditions.

4.3. Projected Gradient Method

As we have seen in section 4.1 we can use any gradient based optimization method to determine an update δu_k if we have obtained (approximate) solutions y_k and λ_k of the state and adjoint equation, respectively. Though there are limitations if box-constrains of the type

$$a \leq u \leq b \tag{4.41}$$

are considered. We then define the feasible set Θ as

$$\Theta := \{x \in \mathbb{R}^K : a \leq x_i \leq b : a,b \in \mathbb{R}\}. \tag{4.42}$$

One could use for example *augmented Lagrangian methods* where the Lagrange function is augmented with some penalty term to form an unconstrained problem, but in this work we restrict ourselves to an extension of the line search method, namely the *Projected Gradient method* [CM87]. We assume here that the Lagrangian L defined in equation (4.6) is convex. The standard line search methods computes a search direction and then decides how far to

move along that direction. The iteration is given by

$$u_{k+1} := u_k + \alpha p^k \tag{4.43}$$

where the positive scalar α is called step length. p_k is the search direction or rather the descent direction and it satisfies the condition

$$(p^k)^T L_u^k < 0, \tag{4.44}$$

with $L_u^k := L_u(y_k^*, u_k, \lambda_k^*)$ and y_k^*, λ_k^* are the solutions of the state and adjoint equation for the given u_k. Then it can be guaranteed that the Lagrangian can be reduced along this direction [NW06]. Typically the steepest descent direction

$$p^k := -L_u^k \tag{4.45}$$

is used that obviously fulfils condition (4.44). Although we assume that the iterand u_k is feasible, i.e. it satisfies the constraints (4.41), it might happen that u_{k+1} defined in equation (4.43) will be infeasible. To overcome this problem, we force the new point $u_k - \alpha N_u^k$ to stay feasible by projecting it onto the feasible set Θ. Thus the iteration for the projected Gradient method is given by

$$u_{k+1} := u_k(\alpha) := P_\Theta(u_k - \alpha L_u^k) \tag{4.46}$$

where P_Ω is a projection onto Θ with

$$(P_\Theta(x))_i := \begin{cases} a, & x_i \leq a \\ x_i, & a < x_i < b \\ b, & x_i \geq b \end{cases} \tag{4.47}$$

Then the projected Gradient p_Θ^k is defined as

$$p_\Theta^k := \frac{1}{\alpha}(u_k(\alpha) - u_k) \tag{4.48}$$

and the iteration in equation (4.48) can be written as

$$u_{k+1} := u_k + \alpha p_\Theta^k. \tag{4.49}$$

Now we are facing the problem of finding an adequate step length α. The ideal choice would be the global minimizer of

$$\Phi(\alpha) := L(y_k(\alpha), u_k(\alpha), \lambda_k(\alpha)), \tag{4.50}$$

where $y_k(\alpha)$ and $\lambda_k(\alpha)$ are again the solutions of the state and adjoint equation for a given $u_k(\alpha)$. But it is too expensive to identify this value. It is more practical to perform an inexact line search to identify a step length that achieves sufficient reduction in L at minimal cost. A popular inexact line search condition stipulates that α should first of all give sufficient decrease in the Lagrangian L. This can be measured by the following inequality:

$$\Phi(\alpha) \leq \Phi(0) + c\alpha(p_\Theta^k)^T L_u^k = \Phi(0) + c(u_k(\alpha) - u_k)^T L_u^k \tag{4.51}$$

with a constant $c \in (0,1)$ and $\Phi(0) := L(y_k, u_k, \lambda_k)$. In the case of standard line search inequality (4.51) is called *Armijo condition*. Furthermore we want to ensure that the step length α is not too short. To ensure this we use the inequality

$$\Phi(\alpha) \geq \Phi(0) + (1-c)(u_k(\alpha) - u_k)^T L_u^k. \tag{4.52}$$

Calamai and More [CM87] showed that, if $\{u_k\}$ is a sequence generated by equation (4.46) with α satisfying the inequalities (4.51) and (4.52), then the limit point u_* of that sequence is a stationary point of L for $u \in \Omega$ in the sense that it satisfies the variational inequality

$$(u - u_*)^T L_u(y_*, u_*, \lambda_*) \geq 0, \quad \forall u \in \Omega. \tag{4.53}$$

To utilize this as a convergence criterion, we assume that we have a candidate $u_{\bar{k}}$ for a limit point such that hopefully $u_* = u_{\bar{k}}$. In order for this to be true, for the next update $u_{\bar{k}+1}$ inequality (4.53) has to hold:

$$(u_{\bar{k}+1} - u_{\bar{k}})^T L_u^{\bar{k}} = \alpha(p_\Theta^{\bar{k}})^T L_u^{\bar{k}} \geq 0 \Leftrightarrow (p_\Theta^{\bar{k}})^T L_u^{\bar{k}} \geq 0 \tag{4.54}$$

We have to consider two cases. First, if $L_u^{\bar{k}} = 0$ then this inequality is obviously true and $u_{\bar{k}}$ is a stationary point. This would be the case if no constraints are active at this point. If $L_u^{\bar{k}} \neq 0$, then we must have

$$p_\Theta^{\bar{k}} = 0, \tag{4.55}$$

for $u_{\bar{k}}$ being a stationary point. Otherwise, if $u_{\bar{k}}$ is not a stationary point, then $(p_\Theta^{\bar{k}})^T L_u^{\bar{k}} < 0$ because $p_\Theta^{\bar{k}}$ is a descent direction [CM87]. Note, that in the first case also equation (4.55) holds since then $p_\Theta^{\bar{k}} = -L_u^{\bar{k}}$. We can thus stop our iteration at an index $\bar{k}$ if $\|p_\Theta^{\bar{k}}\|$ is below some threshold.

Chapter 5.

Numerical Results

This chapter presents several numerical results using the discrete adjoint method introduced in the preceding part of this work. The main focus will be the behaviour of the different ways of calculating the adjoint variable discussed in section 4.2. Prior to that we will validate the gradients calculated with algorithmic differentiation.

For the numerical results we consider the following tracking-type model problem on $\Omega = (0,1)^2$ with boundary and distributed controls:

$$\min J(y,u,v) := \frac{1}{2}\|y - y_\Omega\|^2_{L^2(\Omega)} + \frac{c_1}{2}\|v\|^2_{L^2(\Omega)} + \frac{c_2}{2}\|u\|^2_{L^2(\partial\Omega)} \tag{5.1}$$

subject to

$$\left.\begin{array}{rcll} -\Delta y + y + e^y &=& v & \text{in } \Omega \\ \partial_n y + y^4 &=& u^4 & \text{on } \partial\Omega \\ 0 \leq u(x) \leq 1 & & & \\ -1 \leq v(x) \leq 1 & & & \end{array}\right\} \tag{5.2}$$

The target function $y_\Omega \in L^2(\Omega)$ (see figure 5.1) is defined as

$$y_\Omega(x) = \begin{cases} (2x_1 - 1)^2(2x_2 - 1)^2, & if\ (x_1, x_2) \in [0, \frac{1}{2}]^2, \\ 0, & otherwise, \end{cases} \tag{5.3}$$

where $x := (x_1, x_2)^T$. The bounds for u and v are defined pointwise. $c_1, c_2 \in \mathbb{R}$ are regularization parameter. For all computations a mesh with 256 elements and a polynomial order of 2 is used. This results in a total of 2304 degrees of freedom. The accuracy of the linear solver is set to 10^{-16}. For the discretization we split the vector $\tilde{u}$ defined in (3.49) into parts coming from the discretization of u and v, which we call $\tilde{u} \in \mathbb{R}^U$ and $\tilde{v} \in \mathbb{R}^N$, respectively. As a result all discrete functions are now considered to be additionally dependent on the vector $\tilde{v}$.

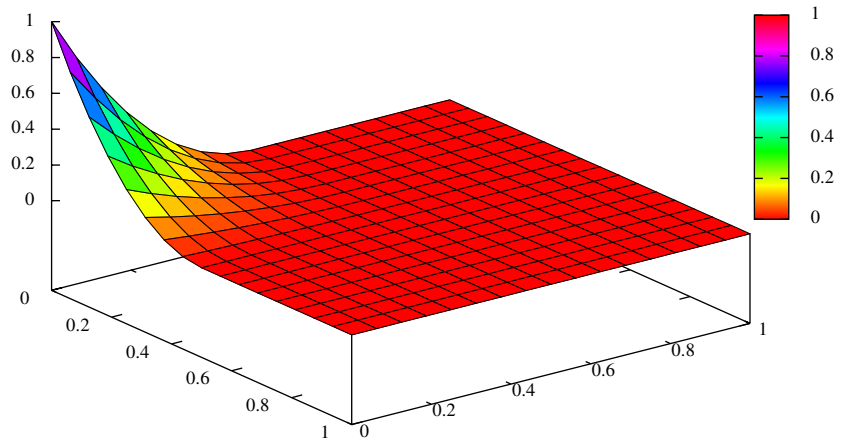


Figure 5.1.: Plot of the target function y_Ω

5.1. Gradient Validation

Since all the needed gradients are calculated more or less implicitly during the execution of the program, it is important to make sure that they contain the correct information. Therefore, we will compare them to the finite difference method. This way of calculating a gradient of a function can be seen as an approximation of the forward mode. Assume we have a function $f(x) : \mathbb{R}^n \to \mathbb{R}^m$. Recall from section 2.1 that applying the forward mode yields

$$\dot{r} = \frac{df}{dx}(x) \cdot \dot{x} \tag{5.4}$$

for given $\dot{x} \in \mathbb{R}^n$. Directional derivatives can be approximated with forward differences, thus we have

$$\dot{r} \approx \dot{r}_{fd} := \frac{f(x + h\dot{x}) - f(x)}{h} \tag{5.5}$$

for some $h \in \mathbb{R}$. The quality of the approximation is mainly influenced by the choice of h that may be a matter of trial and error due to numerical effects resulting from floating-point arithmetic. Hence it makes sense to consider the value $\|\dot{r}_{fd} - \dot{r}\|_2$ or $|\dot{r}_{fd} - \dot{r}|$ if $m = 1$ for different values of h.

First we want to validate the Jacobian of the nonlinear system, i.e. the gradient $F_y := \frac{\partial F}{\partial \tilde{y}}$, where F is defined by equation (3.51). Therefore, we neglect the dependency of F from $\tilde{u}, \tilde{v}$ by considering $\tilde{u} = (0.5, 0.5, \ldots, 0.5)^T$ and $\tilde{v} = (0.5, 0.5, \ldots, 0.5)^T$ fixed. Then we can calculate $\|\dot{r}_{fd} - \dot{r}\|_2$ with $f := F$ and $x := \tilde{y}$ in equation (5.4) and (5.5). In figure 5.2 this value is plotted against different values for h with $\dot{x} = (1, 1, \ldots, 1)^T$ at $\tilde{y} = (0.5, 0.5, \ldots, 0.5)^T$. For $h = 0.01$ through $h = 10^{-8}$ we observe a constant decrease up to $\|\dot{r}_{fd} - \dot{r}\|_2 \approx 10^{-6}$ which implies that the gradient calculated with finite differences converges to the gradient calculated

with algorithmic differentiation. The diverging behaviour for smaller values of h is typical for finite differences as cancellation errors begin to deteriorate the outcome.

Now we will turn to the derivatives of the discretized functional $\tilde{J} : \mathbb{R}^N \times \mathbb{R}^U \times \mathbb{R}^N \to \mathbb{R}$ with respect to its input arguments $\tilde{y}, \tilde{u}$ and $\tilde{v}$. We denote them by J_y, J_u and J_v, respectively. For the validation we can go through the same steps as we did for F_y. Namely we set $\tilde{y} = \tilde{v} = (0.5, 0.5, \ldots, 0.5)^T$ and $\tilde{u} = (0.5, 0.5, \ldots, 0.5)^T$ and use $f := \tilde{J}$ in equation (5.4) and (5.5) by considering all arguments as fixed except the one that corresponds to the specific gradient we want to validate. Thus, we can compute the value $|\dot{r}_{fd} - \dot{r}|$ for J_y, J_u and J_v if $x := \tilde{y}, \tilde{u}, \tilde{v}$, respectively. Again, we use $\dot{x} = (1, 1, \ldots, 1)^T$. The results are shown in figure 5.3. For J_u and J_v we see again a decrease up to $|\dot{r}_{fd} - \dot{r}| \approx 10^{-8}$ for $h = 10^{-6}$ and a deterioration for smaller h. The behaviour for J_y is quite similar, though the minimal value is not as small as for the other gradients.

At last we consider the gradients of the shifted Lagrangian N. For this validation we can simply repeat the procedure for the gradients of $\tilde{J}$ from the paragraph above with $\tilde{J}$ replaced by N. This yields the results shown in figure 5.4. Note that N inherits G and therefore also products of F_y^{-1} and F, thus N has a strongly nonlinear dependence on $\tilde{y}$ and $\tilde{u}$. This is the main reason why the finite difference approximations of N_y and N_u are worse than the approximation of N_v. It can be said, however, that it is possible to find values for h in a relatively small interval that give much better approximations. Nevertheless, we observe convergence to the gradients calculated with algorithmic differentiation.

5.2. Convergence of the state and adjoint equation

In this section we will have a closer look on the convergence of the adjoint and state equation. Although this is only reasonable for "true" iterative methods like Newton's method, Reverse Accumulation and Piggy Back, we will also consider the iterands of the Two Phase method. For that purpose we perform one step of the Discrete Adjoint method with initial values $u_0(x) = 0.5$ and $v_0(x) = y_0(x) = 0.5$. The index $k = 0$ is omitted in the following. We compare the convergence for two different desired values $\epsilon \in \mathbb{R}$ of the residuals. Therefore, the Newton iteration (ref. equation (3.52)) stops at an index m, if

$$\|F(\tilde{y}^m, \tilde{u}, \tilde{v})\| \leq \epsilon \tag{5.6}$$

holds. If Reverse Accumulation is used, iteration (4.16) stops if

$$\|\lambda^{l-1} - \lambda^l\| \leq \epsilon. \tag{5.7}$$

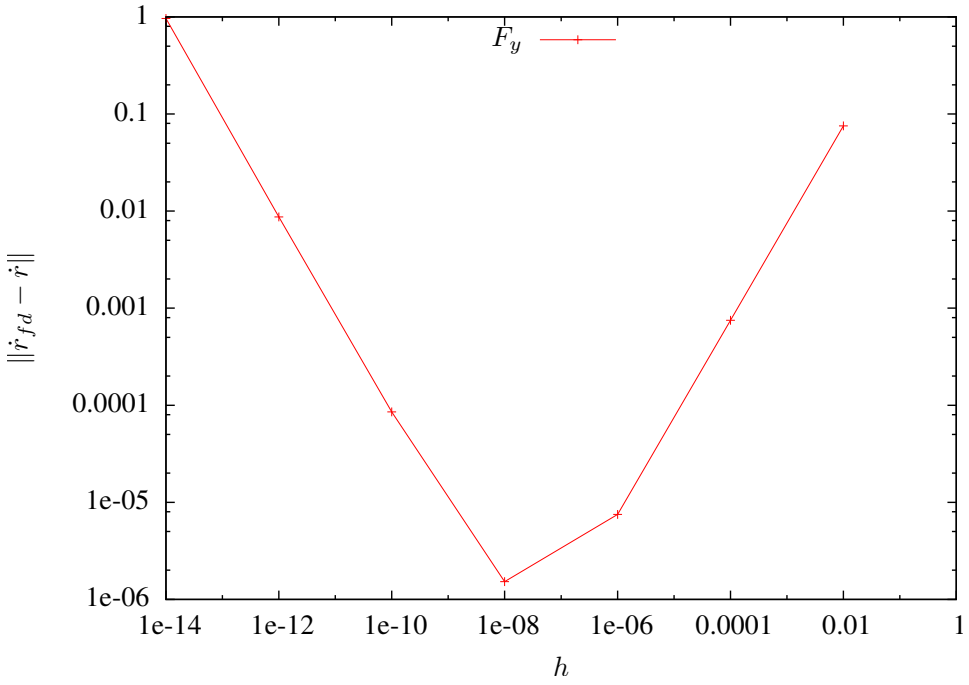


Figure 5.2.: Validation of F_y ($f := F$, $x := \tilde{y}$).

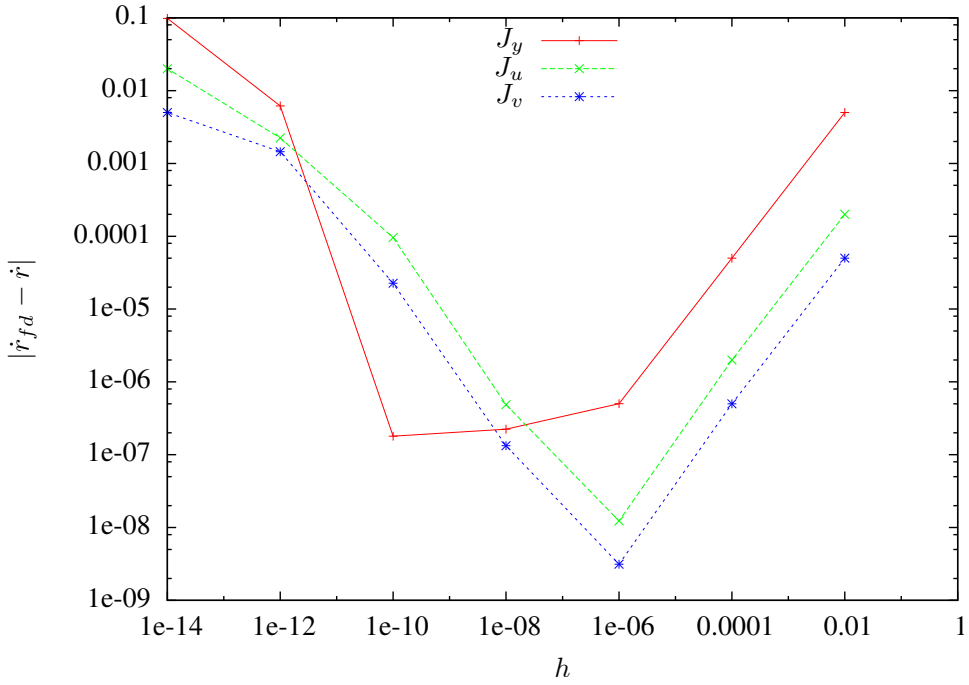


Figure 5.3.: Validation of the derivatives of $\tilde{J}$ ($f := \tilde{J}$, $x := \tilde{y}, \tilde{u}, \tilde{v}$, respectively).

For the Piggy Back method both equations, (5.6) and (5.7), have to be satisfied for the same index $m = l$ in order for iteration (4.18) to stop. In figure 5.5 the residuals $\|F(\tilde{y^m}, \tilde{u}, \tilde{v})\|$ and $\|\lambda^{l-1} - \lambda^l\|$ are plotted against the iteration indices for $\epsilon = 10^{-3}$ and $\epsilon = 10^{-12}$. The trajectory

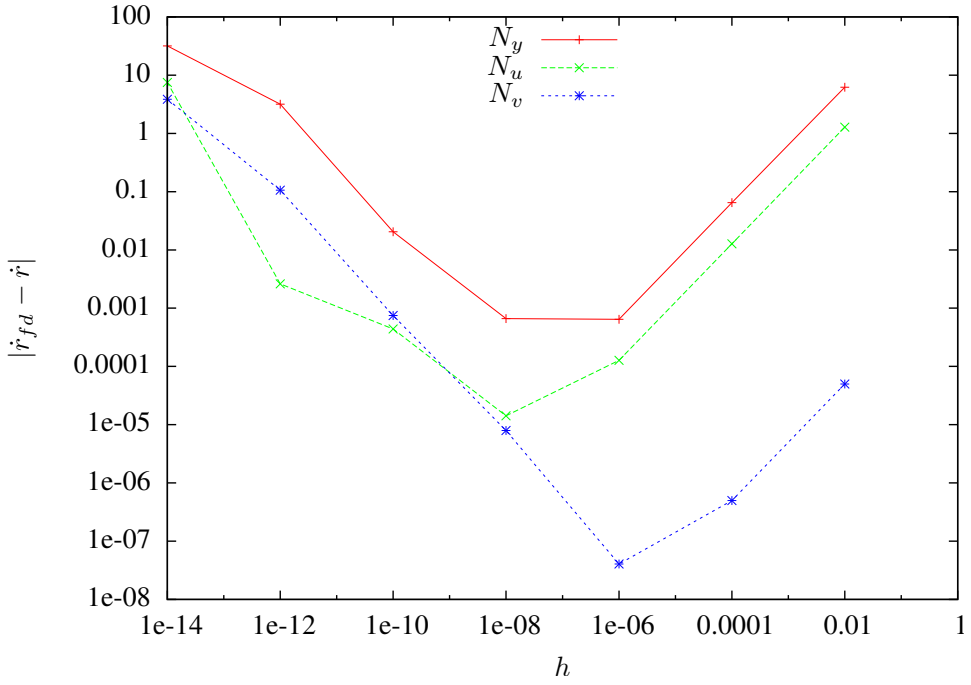


Figure 5.4.: Validation of the derivatives of N ($f := N$, $x := \tilde{y}, \tilde{u}, \tilde{v}$, respectively).

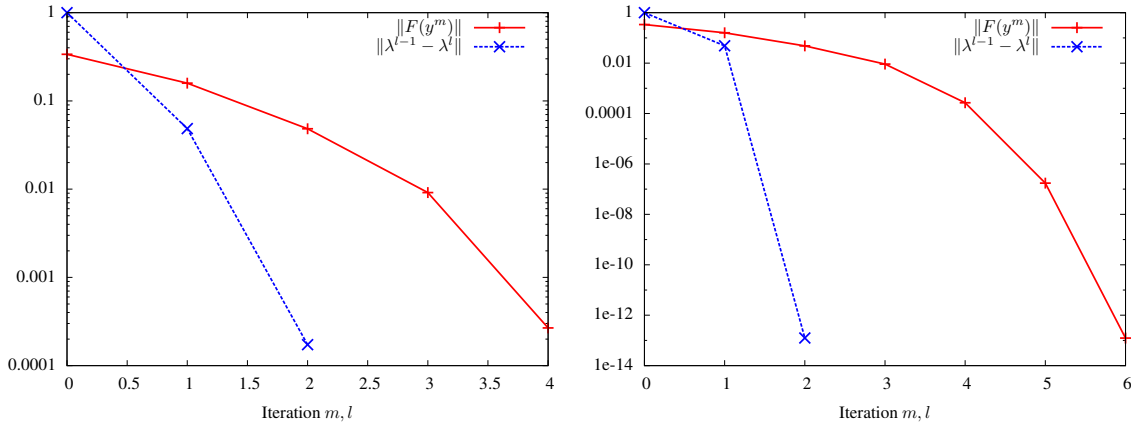
of the Newton iteration (red curve) shows the typical local quadratic order of convergence. Although we would expect a similar convergence for iteration (4.16) (blue curve), in our case it reaches the desired accuracy in just two steps for both values of ϵ. Figure 5.6 shows the same residuals plotted for the coupled iteration (4.18) of the Piggy Back method. Six iterations are necessary to meet the aforementioned stopping criterion for the adjoint (equation (5.7)) for $\epsilon = 10^{-3}$ whereas the residual of the state equation $\|F(\tilde{y}^6, \tilde{u}, \tilde{v})\|$ is already well below 10^{-12}. For $\epsilon = 10^{-12}$ it becomes apparent that the delayed convergence of the adjoint is primarily due to the non-monotonic behaviour of the residual at $m = 2$ through $m = 4$.

As already mentioned for the Two-Phase method there is no residual the we can check to estimate the quality of the approximation of the adjoint. But since λ is defined as the sum of the adjoints of the Newton steps and the discretized functional $\tilde{J}$ (ref. equation (4.30)) it seems reasonable to look at the size of $\bar{y}^m$ to gain some insight into the method. Therefore in figure 5.7 $\|\bar{y}^m\|$ is plotted against the iteration index along with the residuals of the Newton method. For $\epsilon = 10^{-3}$ we observe qualitatively the expected result coming from theory, namely that the values $\|\bar{y}^m\|$ are monotonically increasing with the iteration index. Though they exhibit orders of magnitude ranging from $\|\bar{y}^0\| \approx 8 \times 10^{-5}$ to $\|\bar{y}^4\| \approx 5 \times 10^{-3}$, which means they all have non-negligible influence on the final value of λ. If we increase the accuracy of the Newton solver, i.e. if $\epsilon = 10^{-12}$, the size of $\|\bar{y}^m\|$ for early iterates becomes quite small ($\|\bar{y}^0\| \approx 10^{-13}$) and only the value for $m = \bar{m} = 6$ essentially determines λ. Thus, due to equation (4.30) we

can deduce that we would get for $\epsilon \to 0$ (and consequently for $\|F(\tilde{y}^{\bar{m}}, \tilde{u}, \tilde{v})\| \to 0$):

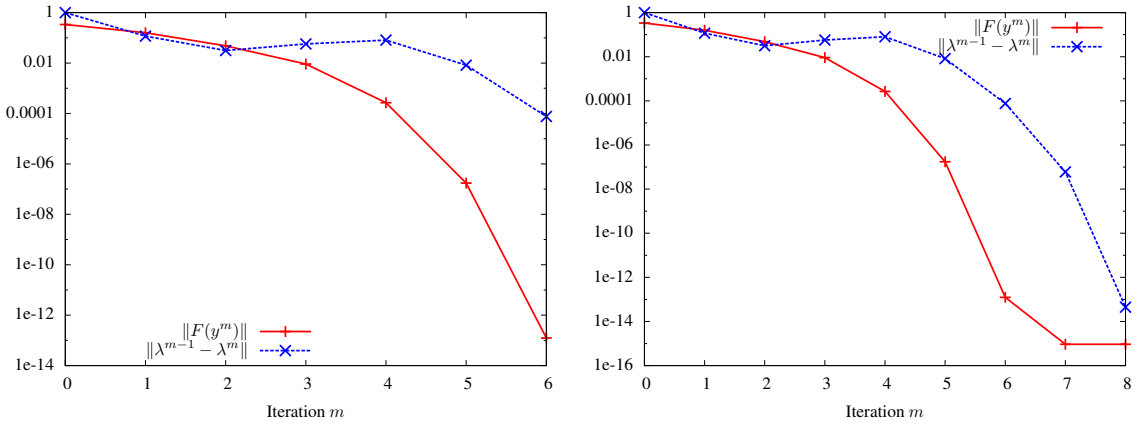
$$\lambda = \tilde{J}_y^T(\tilde{y}^{\bar{m}}, \tilde{u}, \tilde{v}) \tag{5.8}$$

which is nothing but equation (4.40) for determining λ in Giles Exact Dual method.



(a) $\epsilon = 10^{-3}$ (b) $\epsilon = 10^{-12}$

Figure 5.5.: Reverse Accumulation: Convergence of the state and adjoint equation

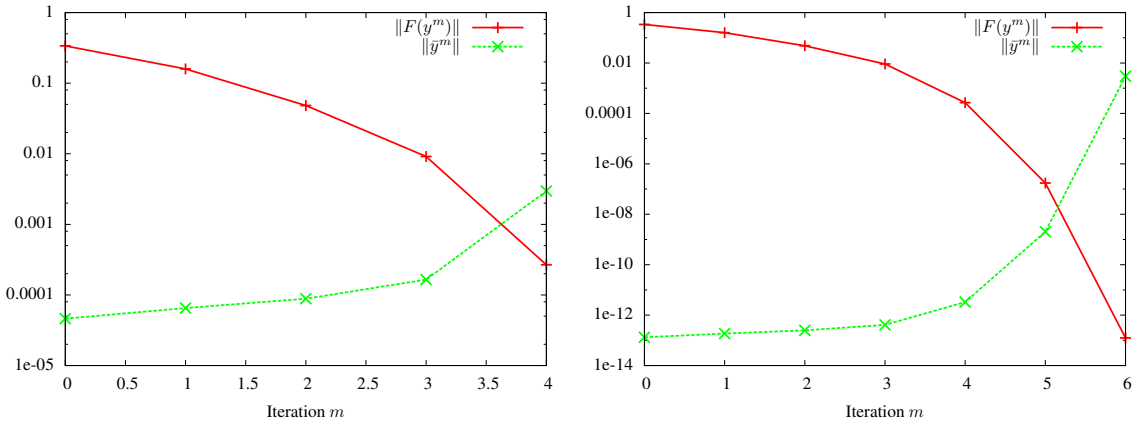


(a) $\epsilon = 10^{-3}$ (b) $\epsilon = 10^{-12}$

Figure 5.6.: Piggy Back: Convergence of the state and adjoint equation

5.3. Convergence of the discrete Adjoint Method

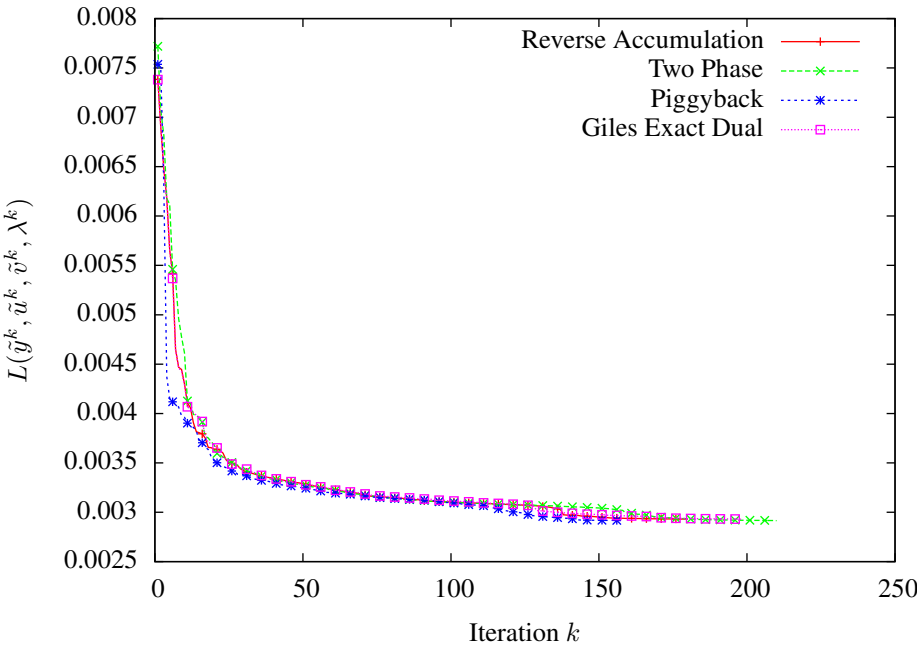
Now we apply the discrete Adjoint method (see figure 4.1) to actually get a solution of problem (5.1) - (5.2). For the stopping criterion of the projected Gradient method we use $\|p_\Theta^k\| \leq 10^{-5}$.



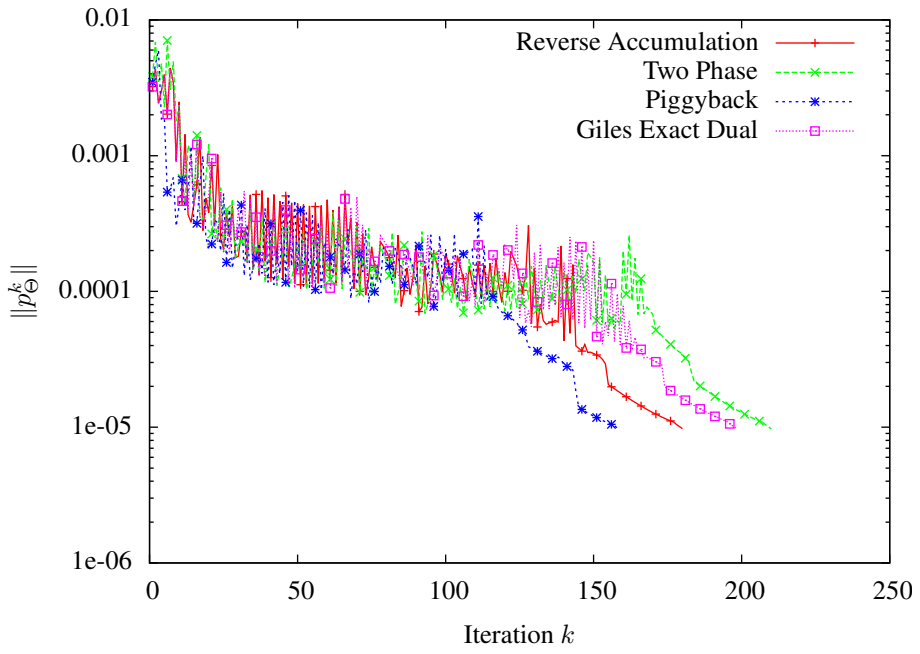
(a) $\epsilon = 10^{-3}$ (b) $\epsilon = 10^{-12}$

Figure 5.7.: Two Phase: Convergence of the state equation and norm of $\bar{y}^m$

As in the previous section we compare the convergence for different accuracies ϵ of the state and adjoint equation. In figure 5.8 the value of the Lagrangian L and the norm of the projected Gradient p_Θ^k is plotted for $\epsilon = 10^{-3}$ at each iteration k and using each of the different methods for solving the adjoint equation. It is obvious that we have convergence for all methods, although the runs of the trajectories differ considerably even for early iterates. For large k we observe a region were the norm of the projected Gradient is monotonically decreasing for all methods. The Piggy Back method seems to perform best, as the value of the Lagrangian decreases most rapidly and the norm of the projected Gradient reaches the desired accuracy within the minimal number of iterations. But due to the observations we made in the last section we can conclude that this is mainly because of the much better approximation of the state variable $\tilde{y}_k$ in each iteration. If the accuracy is increased, i.e. $\epsilon = 10^{-6}$, shown in figure 5.9, there is almost no difference in either of the trajectories up to $k = 20$. For the Two-Phase method the gradient exhibits a more oscillatory behaviour beginning at that point. The Piggy Back method converges within the same number of iterations as the Two-Phase method, so the higher accuracy of $\tilde{y}_k$ is obviously no advantage anymore. Interestingly Giles Exact Dual now requires the least amount of iterations. For $\epsilon = 10^{-12}$ we observe, as expected, smaller differences between the methods, although the Two Phase method still begins to differ quite early. Nevertheless, they all require almost the same number of iterations to reach the stopping criterion, though the increased accuracy does not lead to faster convergence as one might expect.

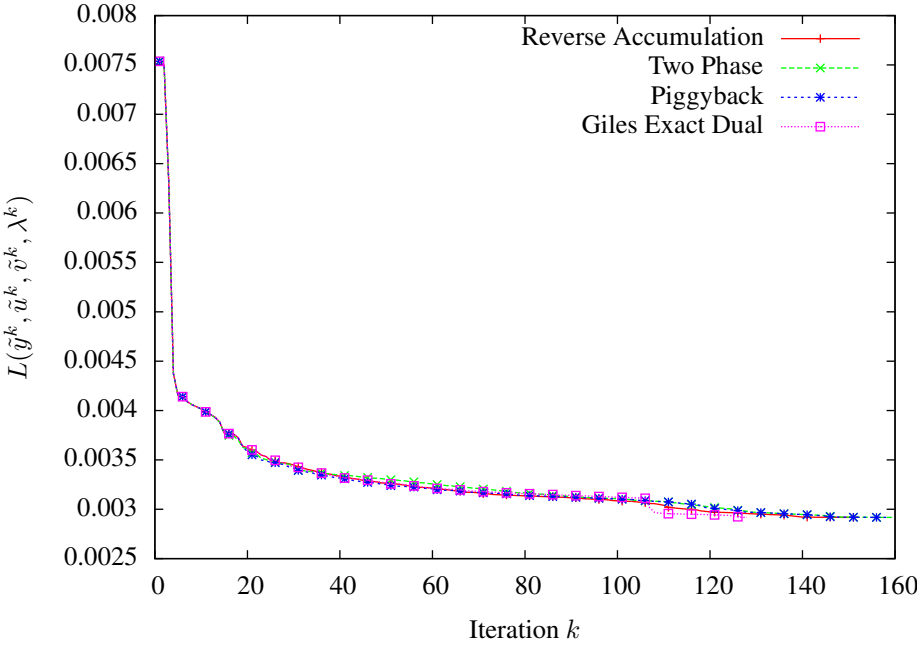


(a) Value of the Lagrangian L

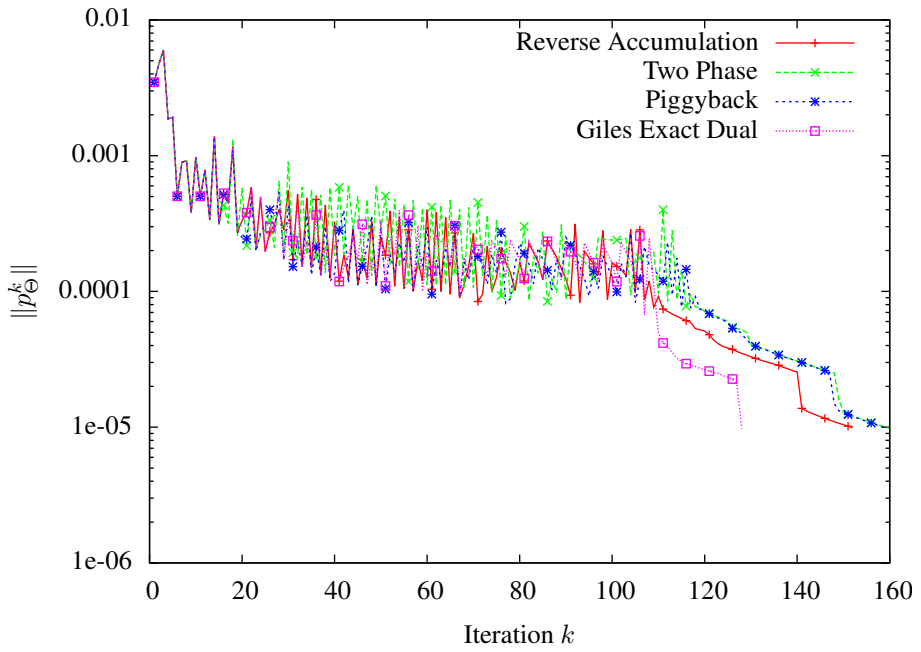


(b) Norm of the projected Gradient p_Θ^k

Figure 5.8.: Convergence of the discrete Adjoint method for $\epsilon = 10^{-3}$.

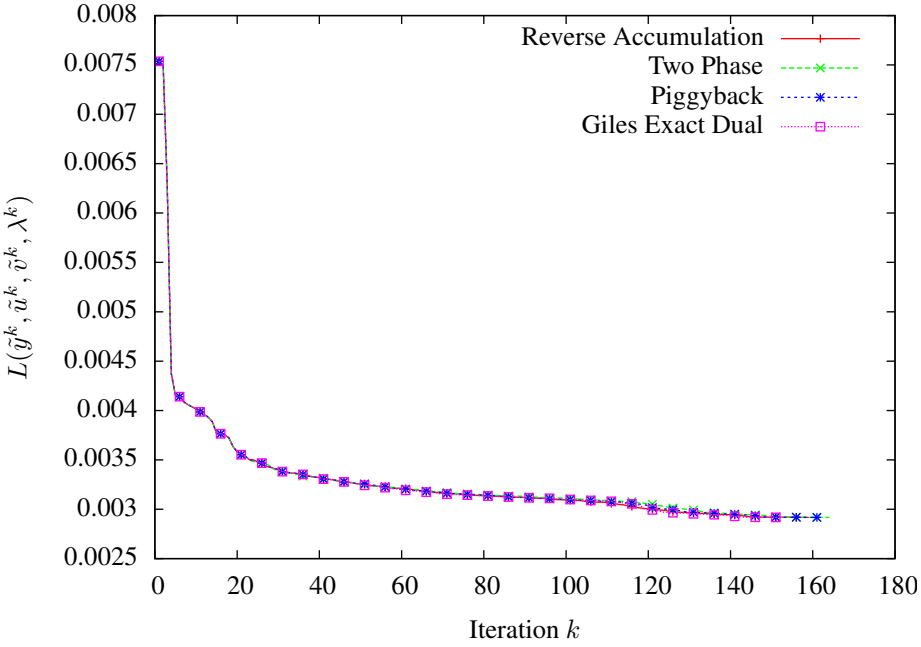


(a) Value of the Lagrangian L

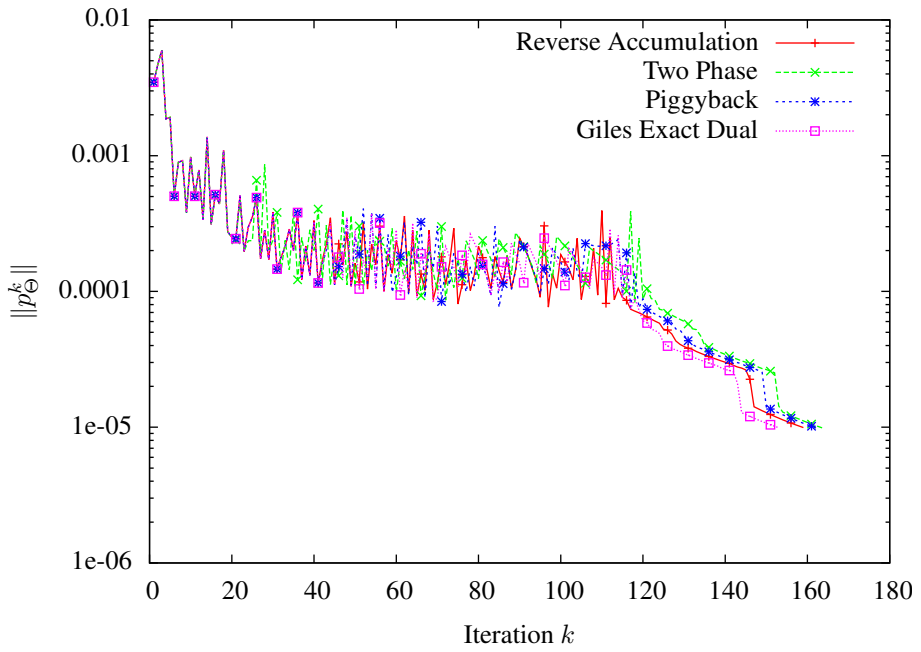


(b) Norm of the projected Gradient p_Θ^k

Figure 5.9.: Convergence of the discrete Adjoint method for $\epsilon = 10^{-6}$.



(a) Value of the Lagrangian L



(b) Norm of the projected Gradient p_Θ^k

Figure 5.10.: Convergence of the discrete Adjoint method for $\epsilon = 10^{-12}$.

Chapter 6.

Conclusion

In this work we introduced several methods for solving the adjoint equation, and investigated the convergence of the iterative ones regarding their performance if the state equation is solved with different accuracies. Especially for the Two-Phase method it was found that for low accuracies information from each step of the fixed-point iteration has measurable influence on the adjoint. Reverse Accumulation showed very fast convergence whereas for the Piggy Back method the convergence of the adjoint lagged massively behind the convergence of the state variable. Furthermore, we have seen in the last chapter how the discrete Adjoint method behaves if the state and adjoint equations are solved with low accuracy. Although one would expect that especially Giles Exact Dual would fail to converge because of its rather unrealistic assumption, we showed that all methods for solving the adjoint equation work quite well in terms of convergence. Contrary to intuition it was further noticed that higher accuracy does not necessarily means faster convergence. There are two possible reasons why these methods offer quite similar convergence results in our case:

1. The projected Gradient method, which is essentially a steepest Descent Line Search, is very insensitive to errors in the gradient. As long as the Lagrangian can be reduced somewhere along the search direction, we observe convergence.

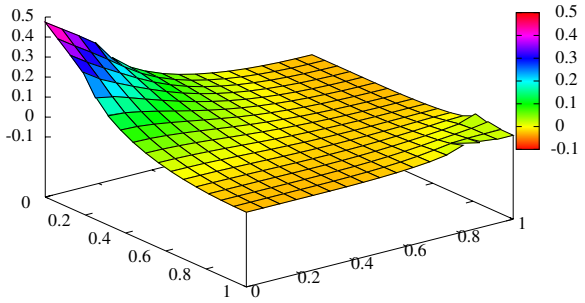
2. The non-linear Poisson equation is a fairly simple mathematical model where no singularities can occur in the solution. Which means all gradients and higher derivatives are fairly smooth.

Regarding point 1 it would be interesting to investigate how the methods perform if we incorporate second order information for finding an update of the control. For example future work could be to implement a Quasi-Newton method instead of the projected Gradient method. For point 2 the model equation could be changed to a pure convection or convection-diffusion problem.

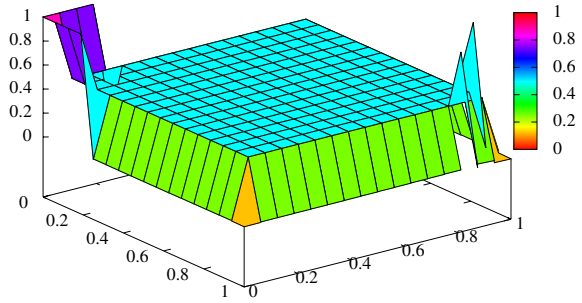
Appendix A.

Example Solutions

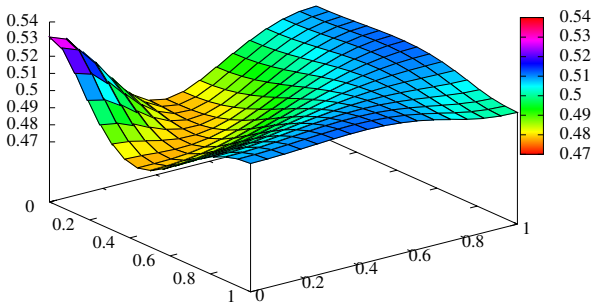
Since for all the methods the discrete Adjoint iteration did converge the calculated solutions do not differ much. Therefore, we present here only a few plots showing the solutions of the optimal control problem (5.1) - (5.2).



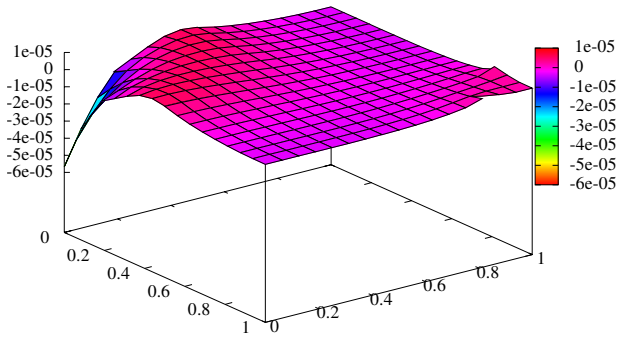
(a) State y



(b) Boundary Control u

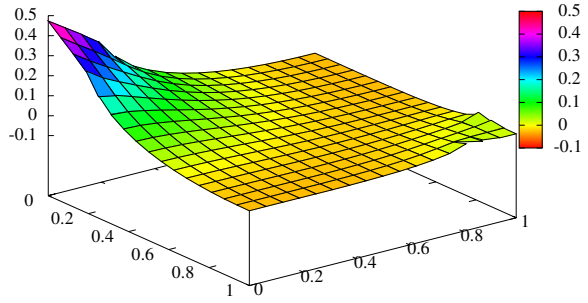


(c) Distributed Control v

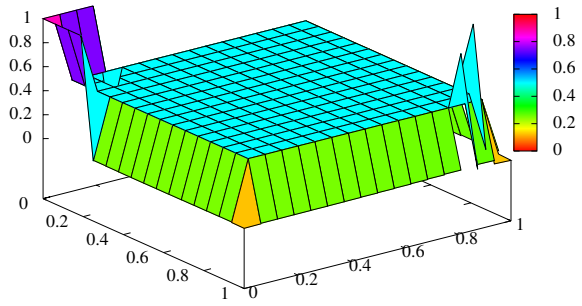


(d) Adjoint λ

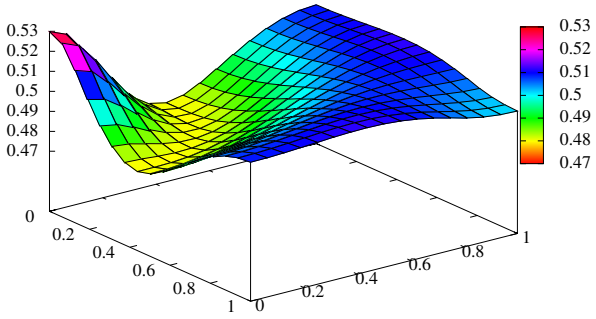
Figure A.1.: Two Phase: Plot of the solution for $\epsilon = 10^{-3}$.



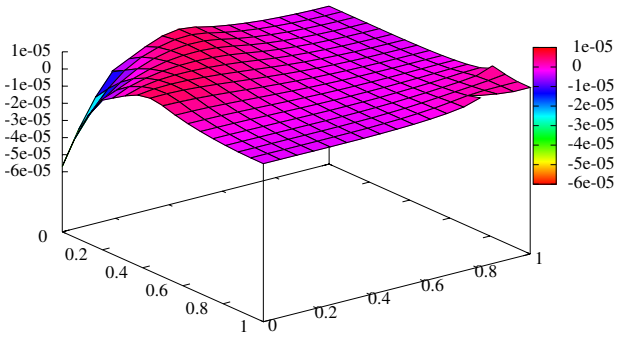
(a) State y



(b) Boundary Control u



(c) Distributed Control v



(d) Adjoint λ

Figure A.2.: Two Phase: Plot of the solution for $\epsilon = 10^{-12}$.

Bibliography

[BG05] Rommel Bustinza and Gabriel N. Gatica. A local discontinuous galerkin method for nonlinear diffusion problems with mixed boundary conditions. *SIAM J. Sci. Comput.*, 26(1):152–177, January 2005.

[BHK07] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II – a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4):24/1–24/27, 2007.

[Chr98] Bruce Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

[CM87] Paul H. Calamai and Jorge J. More:9A. Projected gradient methods for linearly constrained problems. *Math. Program.*, 39(1):93–116, October 1987.

[DPE11] D.A. Di Pietro and A. Ern. *Mathematical Aspects of Discontinuous Galerkin Methods*. Mathématiques et Applications. Springer, 2011.

[DR08] Wolfgang Dahmen and Arnold Reusken. *Numerik für Ingenieure und Naturwissenschaftler* -. Springer DE, Berlin, 2. korr. aufl. edition, 2008.

[GP00] Michael B. Giles and Niles A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65:393–415, 2000.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

[Har08] Ralf Hartmann. Numerical analysis of higher order discontinuous Galerkin finite element methods. In H. Deconinck, editor, *VKI LS 2008-08: CFD - ADIGMA course on very high order discretization methods, Oct. 13-17, 2008*. Von Karman Institute for Fluid Dynamics, Rhode Saint Genèse, Belgium, 2008.

[LLN12] Johannes Lotz, Klaus Leppkes, and Uwe Naumann. dco/c++ - derivative code by overloading in c++. Technical Report AIB-2011-06, RWTH Aachen, May 2012.

[NW06] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[OR70] J.M.A. Ortega and W.C.A. Rheinboldt. *Iterative Solution of Non Linear Equations in Several Variables*. Computer science and applied mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1970.

[SS86] Youcef Saad and Martin H Schultz. Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, July 1986.

[Trö10] F. Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods, and Applications*. Graduate Studies in Mathematics. American Mathematical Society, 2010.