Towards Pseudospectral Control and Estimation

Reuterswärd, Philip

Towards Pseudospectral Control and Estimation

Philip Reutersward

Department of Automatic Control
Lund University
Lund, December 2012

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

ISSN 0280–5316
ISRN LUTFD2/TFRT--3257--SE

Errata

p.15 Line 11: "...problems of the type

$$\min_{u} \int_{t_0}^{t_1} g(x,\, u)\, dt.$$

"

p.16 Line 2: "...and sampled output

$$\min_{\hat{x}} \int_{I} \|h(\hat{x}(t)) - y(t)\|^2\, dt$$

"

p.18 Line 14: "...the directional derivative vanishes at an extrema

$$\nabla f(z^*) = \mathbf{0}.$$

"

p.44 The theorem is adapted from [Gong *et al.*, 2008].

p.46 The theorem is adapted from [Gong *et al.*, 2008].

p.46 Line 6: "...the sequence $\{(\dot{x}^N(t),\, u^N(t))\}$."

p.59 Line 5: "…up with

$$
\begin{aligned}
&\begin{bmatrix}
F_1 & & & & & & & & \\
& F_2 & & & & F_2'' & & & \\
& & \ddots & & & & \ddots & & \\
& & & F_{N-1} & & & & F_{N-1}'' & \\
& & & & F_N & & & & F_N'' \\
F_1' & & & & & -I & & & \\
& F_2' & & & & & \ddots & & \\
& & \ddots & & & & & \ddots & \\
& & & F_{N-1}' & & & & & -I
\end{bmatrix}
\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{N-1} \\ z_N \\ d_1 \\ \vdots \\ \vdots \\ d_{N-1} \end{bmatrix} = \\
&\begin{bmatrix}
F_1 z_1 \\
F_2 z_2 + F_2'' d_1 \\
\vdots \\
F_{N-1} z_{N-1} + F_{N-1}'' d_{N-2} \\
F_N z_N + F_N'' d_{N-1} \\
F_1' z_1 - d_1 \\
\vdots \\
F_{N-1}' z_{N-1} - d_{N-1}
\end{bmatrix}
=
\begin{bmatrix} g \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.
\end{aligned}
$$

"

p.66 Item 3: "An optimal control is then solved using $[x(s),\, x(s)+H]$ as initial value."

p.73 Line 8: "…Consider solving

$$\min_{\hat{x}} \int_{T-T_h}^{T} \|h(\hat{x}(t)) - y(t)\|^2 \, dt$$

"

p.73 Line 11: "…The discrete counterpart becomes

$$\min_{\hat{x}} \sum_{k=0}^{N} w_k \|h(\hat{x}^{(k)}) - y^{(k)}\|^2$$

"

p.73 Footnote should be removed.

p.75 Line 6: "…water tank example

$$\begin{cases} \dot{x}_1 = -a_1\sqrt{x}_1 + bu \\ \dot{x}_2 = a_1\sqrt{x}_1 - a_2\sqrt{x}_2 \end{cases} .$$

"

p.89 Author is Fornberg, not Fornbern.

Lukas 17:32

Abstract

This thesis covers different topics related to the application of pseudospectral optimization methods in the field of automatic control. Pseudospectral optimization methods solve dynamic optimization problems by discretizing the state-space, creating a discretized version of a continuous problem. The resulting discretized optimization problems are solved by standard software for nonlinear optimization.

An evaluation of pseudospectral optimal control in a model predictive control (MPC) setting is presented, as a double-tank process is controlled from one set point to another. The method quite often experiences divergence, which makes its use in a real setting limited.

The thesis proposes a way to use pseudospectral optimization as a nonlinear state estimator together with Out-Of-Sequence-Measurements (OOSM). The main idea is outlined; however, it represents future work as a lot remains to be done.

The thesis also evaluates possible performance gains when solving optimization problems governed by ODE system dynamics in parallel. For large systems, with very many discretization points, substantial speed-ups are possible. However, the application is shown only on randomly generated systems, as real world examples of such large sizes are elusive.

Acknowledgments

I thank God, Sweden, the Department of Automatic Control, my supervisor Karl-Erik Årzén, Anders Rantzer and you.

Contents

1 Introduction

Optimization-based methods in industry gain more and more interest as there is potentially a lot of money to gain. Also the word ‘optimal’ has in recent years become a catch-phrase which by itself seems to generate interest [Kang and Bedrossian, 2007]. Optimization can be used; for example, to minimize the resources required in production and thus the money being spent. Examples include grade-changes in large chemical plants; that is, going from production of one product to another using the same plant, or the computation of optimal trajectories in space-craft applications. In this thesis we look at pseudospectral methods in the field of optimal control. Related to the pseudospectral methods are the so called direct methods for dynamic optimization, and we will look at the parallelization of such methods.

1.1 Dynamic Optimization at a Glance

Pseudospectral optimization at its core deals with minimizing an integral cost function

$$\int_{t_0}^{t_1} g(x,\, u)\, dt,$$

for some system dynamics described by an ODE

$$\dot{x} = f(x,\, u); \tag{1.1}$$

for example, with prescribed initial condition $x(t_0) = x_0$. The control input u yields a state trajectory x.

There are ways to solve such problems analytically, but this gets complicated as the complexity of the functions f and g increases. Complexity also increases with different end point constraints; for example, on the state at the final time $x(t_1) = x_1$. Specifying general path constraints

$$h(x, u) \leq 0,$$

makes it even harder.

These problems may be overcome by calculating a numerical approximation using a computer. This way the time interval $[t_0, t_1]$ is divided into a finite set of points. By numerical quadrature the value of the cost function can be calculated, up to a given accuracy. To link the states at different time points, analytically described by the ODE (1.1), methods taken directly from or inspired by the vast field of numerical solution of ODEs are used.

By discretization the continuous problem can now be put in the form

$$\begin{aligned} &\min_{z \in \mathbb{R}^n} f(z) \\ &F_1(z) \leq \mathbf{0} \\ &F_2(z) = \mathbf{0}. \end{aligned}$$

This is a general nonconvex optimization problem that we can solve using numerical algorithms. They work iteratively, in each step getting closer and closer to the true solution usually using some sort of Newton type iteration. Most of the time spent during optimization is spent solving a linear system, stemming from the KKT conditions of the optimization problem.

However, for this to work we need *convergence*; that is, when the grid gets finer the numerical solution should get increasingly closer to the true analytic solution. Depending on the problem, this can not always be guaranteed.

1.2 Pseudospectral Control Applications

The pseudospectral approach is one way to discretize the optimal control problems just described. It gives high accuracy of approximation with a low number of grid points. This reduces the computation time, since we are able to use a coarser discretization while still retaining a high enough solution accuracy, as compared to other methods.

The pseudospectral methods are quite new at least in the control community. However, the theory behind them is relatively well developed. It boils down to checking to see if the optimization converges, assuming some system regularity conditions. If it does it must have converged to the true solution. However, the theory says only what happens if the computation converges, it does not specify how often it does. This is highly dependent on the initial values, the cost function and system dynamics.

Pseudospectral optimization allows us to solve problems of the type

$$\int_{t_0}^{t_1} g(x,\, u)\, dt$$
$$\dot{x} = f(x,\, u),$$

where the cost function is taken to minimize some true system quantity like energy or time. However, from a feedback control perspective this is not really desirable as this implies open-loop control. No feedback is used, we solely depend on model accuracy as the optimization constructs a control trajectory $u(t)$ that produces $x(t)$ given exact system dynamics.

Given a fast and reliable dynamic optimizer we could construct a feedback control by sampling the system and solving a sequence of optimization problems. We shall later try an MPC type scheme where the optimizer runs and then the computation is used to construct a control input valid until the next one is calculated.

We shall also try the pseudospectral optimizer as a nonlinear observer, where we instead look at an output system

$$\begin{aligned}\dot{x} &= f(x(t),\, u(t),\, t)\\ y &= h(x(t)).\end{aligned}$$

Here we wish to find the state of the system, x, by observing the output y. The right hand side of the ODE wears a hat to distinguish it from (1.1). In this context we can limit the feasible trajectories through prior information, since we usually have some knowledge of the system state. This is described by

$$r(\hat{x}(t)) \leq \mathbf{0},$$

and we use $\hat{x}$ to denote the observed system state. The optimizer now minimizes the difference between the simulated and sampled output

$$\begin{aligned} \min_u \int_I \|h(\hat{x}(t)) - y(t)\|^2 \, dt \\ \dot{\hat{x}} = f(\hat{x}(t),\, u(t)\, t) \\ r(\hat{x}(t)) \leq \mathbf{0}, \end{aligned}$$

over some time interval I. We propose to use this in the emerging field of sensor fusion and out-of-sequence measurements, where the pseudospectral optimizer may be viewed as a sensor with delay, the delay being the computation time.

1.3 Contribution

This is the collection of some of my thoughts and experiences using pseudospectral-based and similar optimization methods in the field of automatic control.

I have evaluated pseudospectral optimal control in an model predictive control (MPC) setting. I propose a way to use pseudospectral optimization as a nonlinear state estimator together with Out-Of-Sequence-Measurements (OOSM). This is the embryo of an idea, out of which I believe something later could be developed. I have also evaluated possible performance gains when solving optimization problems governed by ODE system dynamics in parallel.

I have also implemented a programming framework for running the optimizations presented in this thesis. The code is not included in the thesis, although it is used for all presented computations. The overall structure of my approach to solving pseudospectral optimization problems in MATLAB is descried in an appendix.

1.4 Outline

The outline of the thesis is as follows. In Chapter 2 we introduce optimal control. Chapter 3 solves an example problem by discretization, and

Chapter 4 discusses nonconvex optimization. Discretizations are discussed in Chapter 5, and the theory of pseudospectral optimization is given in Chapter 6. In Chapter 7 we look at various ways to numerically solve nonconvex problems. Chapter 8 tests pseudospectral optimal control in an MPC type scheme. In Chapter 9 we look at observers in pseudospectral setting, and finally we try to reduce the computation time by solving dynamic optimization problems in parallel in Chapter 10. Thoughts on implementing a pseudospectral optimizer are found in appendix A, along with code snippets in MATLAB and C.

2

Optimal Control

The term optimal control is very broad and typically people from different fields mean different things. Here we give our definition, and the problems presented are the ones that we later seek numerical solutions to. We shall speak of analytic solutions; however, with nonlinear system dynamics it becomes impossible to solve or represent the solutions in a tractable way. We start by looking at variational calculus, as it is needed for the presentation.

2.1 Variational Calculus

Variational calculus is loosely speaking the differential calculus of functions, see for example [Arnold, 1978]. Differential calculus deals with functions of real numbers, here we consider functions of functions. In optimal control we seek the extremals of *functionals*, were the solution is a function. A functional is any mapping from the space of curves to the real numbers.

For functions defined on $\mathbb{R}^n$ the directional derivative vanishes at an extrema

$$\nabla f(z) = \mathbf{0}, \quad \forall z \in \mathbb{R}^n.$$

Now instead we look at functionals $J : Z \mapsto \mathbb{R}$, where Z is the space of functions under consideration. Here the *Gateaux variation*

$$\delta J(z) := \lim_{\epsilon \to 0} \frac{J(z + \epsilon\delta) - J(z)}{\epsilon} = \frac{\partial}{\partial \epsilon} J(z + \epsilon\delta)\Big|_{\epsilon=0}, \quad z \in Z, \qquad (2.1)$$

at an extrema must be zero; that is,

$$\frac{\partial}{\partial \epsilon} J(z + \epsilon h)\Big|_{\epsilon=0} = \mathbf{0}.$$

Let us look at the variation of the following functional

$$\Phi(y) := \int_a^b L(y(t),\, \dot{y}(t))\, dt, \tag{2.2}$$

which for a given function $[a,\, b] \ni t \mapsto y(t)$ yields a real number. Later we shall construct a function such that it minimizes a given cost functional at the same time fulfilling some system dynamics described by an ODE. According to Definition (2.1) we have

$$\delta\Phi(y) = \frac{\partial}{\partial \epsilon} \int_a^b L(y + \epsilon\delta,\, \dot{y} + \epsilon\dot{\delta})\, dt\Big|_{\epsilon=0} = \int_a^b \frac{\partial L}{\partial y}\delta + \frac{\partial L}{\partial \dot{y}}\dot{\delta}\, dt = \frac{\partial L}{\partial \dot{y}}\delta\Big|_{t=a}^{b} + \int_a^b \Big(\frac{\partial L}{\partial y} - \frac{\partial}{\partial t}\Big(\frac{\partial L}{\partial \dot{y}}\Big)\Big)\delta\, dt,$$

where we have used integration by parts. [1] The boundary term is zero if we assume the value of L to be fixed at the boundary; that is,

$$\frac{\partial L}{\partial \dot{y}}\delta\Big|_{t=a}^{b} = \mathbf{0}.$$

Summarizing we have that

$$\delta\Phi(y) = \int_a^b \Big(\frac{\partial L}{\partial y} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}}\Big)\delta\, dt,$$

must be zero for the variation to vanish. This is true if

$$\frac{\partial L}{\partial y} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}} = \mathbf{0},$$

which is referred to as the Euler-Lagrange equation [Arnold, 1978].

[1]This assumes that the appearing functions are smooth enough.

2.2 An Optimal Control Problem

By optimal control we mean to minimize a functional while at the same time adhering to some system dynamics. A lot of variations exist, let us look here at a problem that prescribes initial and final states

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \int_{t_0}^{t_f} g(x(t),\, u(t))\, dt && (2.3) \\ \text{subject to} \quad & \dot{x} = f(x,\, u) && (2.4) \\ & x(t_0) = x_0, \quad x(t_f) = x_1. && (2.5) \end{aligned}$$

The solution is a trajectory starting at x_0 and ending at x_1 subject to the system dynamics $\dot{x} = f(x,\, u)$. We seek the control input u that makes x follow the ODE and at the same time minimizes the cost assigned to the trajectory by the cost function.

The solution we find is the control input u that we need to make the system follow the optimal path. However, planning the trajectory in advance means open-loop control of the system, something generally avoided by the control community. We shall later look at how to combine this with feedback to make for some interesting control applications.

The solution is optimal with regards to the *cost function*. With different cost functions we usually get different solutions to the optimization problem. This means that the choice of cost function is very important. For example, a constant cost function makes any feasible solution optimal, so care must be exercised.

2.3 Analytic Solution

The survey [Polak, 1973] gives a good overview of optimality conditions for these problems. Let us now look at some conditions that must be fulfilled for a solution to be optimal. To this end we rewrite the cost function

$$\int_{t_0}^{t_f} g(x,\, u)\, dt = \int_{t_0}^{t_f} g(x,\, u) + \lambda^{\mathrm{T}}(f(x,\, u) - \dot{x})\, dt,$$

where we have omitted the time-dependence of x and u for sake of clarity.

This is a functional on the form (2.2), with

$$y = \begin{bmatrix} x \\ u \\ \lambda \end{bmatrix}, \quad L(y,\, \dot{y}) = g(x,\, u) + \lambda^{\mathrm{T}}(f(x,\, u) - \dot{x}).$$

The Euler-Lagrange equation becomes

$$\begin{bmatrix} \frac{\partial L}{\partial x} \\ \frac{\partial L}{\partial u} \\ \frac{\partial L}{\partial \lambda} \end{bmatrix} - \frac{d}{dt} \begin{bmatrix} \frac{\partial L}{\partial \dot{x}} \\ \frac{\partial L}{\partial \dot{u}} \\ \frac{\partial L}{\partial \dot{\lambda}} \end{bmatrix} = \begin{bmatrix} \frac{\partial g}{\partial x} + \frac{\partial}{\partial x}\lambda^{\mathrm{T}} f - \dot{\lambda} \\ \frac{\partial g}{\partial u} + \frac{\partial}{\partial u}\lambda^{\mathrm{T}} f \\ f(x,u) - \dot{x} \end{bmatrix} = \mathbf{0}.$$

If we define the *Hamiltonian* as

$$H(x\,, u\,, \lambda) = g(x,\, u) + \lambda^{\mathrm{T}} f(x,\, u),$$

we can rewrite the Euler-Lagrange equation as

$$\begin{bmatrix} H_x^{\mathrm{T}} - \dot{\lambda} \\ H_u^{\mathrm{T}} \\ f(x,u) - \dot{x} \end{bmatrix} = \mathbf{0}, \tag{2.6}$$

where we use the shorthand

$$H_x := \left[\frac{\partial}{\partial x_1} H \, \dots \, \frac{\partial}{\partial x_n} H \right], \quad H_u := \left[\frac{\partial}{\partial u_1} H \, \dots \, \frac{\partial}{\partial u_p} H \right].$$

The Euler-Lagrange equations (2.6) give conditions for optimality. They state that the ODE holds and gives conditions on the Hamiltonian, valid for an optimal solution to the control problem. From this an analytical solution can be found, however then some extra conditions need be satisfied. For example; if the Hamiltonian is positive definite with respect to the control input; that is

$$\frac{\partial^2}{\partial u^2} H > 0,$$

then the optimal control input can be found as

$$u^* = \arg\min_u H(x^*,\, u,\, \lambda^*).$$

This is sometimes referred to as the *maximum principle*, see for example [Pontryagin *et al.*, 1962]. Fixing various end point constraints results in variations to this formulation. As the complexity of the problem increases, both in terms of system dynamics and end point constraints, an analytical solution gets harder and harder to find this way (see [Polak, 1973] for a list of variations). For example; incorporating general path inequalities

$$h(x(t), u(t)) \leq 0, \quad t \in [t_0, t_1],$$

is more easily done in a numerical setting.

Hamiltonians are commonly used in conjunction with mechanical systems. There they represent the total energy of a system. Here instead we speak of energy in a more general system, valid outside the laws of physics.

2.4 Numerical Solution

There are various ways to solve optimal control problems numerically. Early methods are surveyed in [Polak, 1973], and more recent developments in [Magni *et al.*, 2009].

When the complexity of the problems increases we tend to look for numerical alternatives. Methods can be classified as *direct* or *indirect*. The former methods work directly on the initial optimization problem (2.3), whereas the latter work with the Euler-Lagrange equations (2.6), as such solving the original optimization problem indirectly.

Indirect Methods

The formulation with a Hamiltonian in the last section naturally leads to Two-Point Boundary Value (TPBV) problems. The values prescribed at the boundaries stem from the various possible end-point conditions in the original optimization formulation.

One way to solve these problems is by means of the *shooting method* [Betts, 2001]. From an initial control input the system is simulated. When the simulation reaches the end-point, we check if the end-point condition is fulfilled. If it is not we refine our initial input and start over. This is repeated until convergence is found. There is also *multiple shooting* [Betts, 2001], where the control interval is divided into sub intervals. The same procedure is now carried out over these sub intervals, and when the end-points of the sub intervals meet, convergence is declared.

Another way to solve the optimization problem is by using *sequential methods* [Betts, 2001]. These approximate the states and controls in the TPBV by polynomials, and then iteratively find the solution.

Direct Methods

The direct methods are conceptually simpler, as they work directly with the original optimization formulation. The last twenty years, so called *simultaneous methods* have gained in popularity. Here the complete state and control trajectories are discretized and the solution over the whole interval is calculated at once using an optimization code. Various discretization styles exist, most of them stemming from the vast field of numerical solution of ODE. Especially *collocation* methods have been popular, due to the high order of numerical accuracy they possess. The pseudospectral method is a direct method and forms the basis of this thesis.

3

An Example

We shall now try to solve an optimal control problem. This is done by discretization and the discretized problem is solved using a numerical algorithm. This is quite straightforward; however, the procedure raises some questions. The choice of discretization seems arbitrary, as anything can be fed to a numerical optimizer. In later chapters we shall look at the underlying theory which will then make clear what choices make sense.

3.1 Problem

The optimization methods we shall consider later are able to deal with nonlinearities, but here we use a linear version to more easily show how the numerical solution proceeds. To this end consider a linear system

$$\dot{x} = Ax + Bu.$$

In a real setting usually we have a limit on the control input, we model the limitations of the actuator as

$$\begin{gathered}\dot{x} = Ax + Bu \\ u_{\min} \leq u \leq u_{\max}.\end{gathered}$$

Now assigning a cost function to the system we arrive at an optimal control problem on the form discussed in previous chapters. The cost function could, for example, penalize the control power. This is the choice here as we consider the problem to minimize the control power of a general linear system going from an initial to a final state in a given time

$$\begin{aligned}\min_{x,\,u} \int_0^T u^2 \\ \dot{x} = Ax + Bu \\ u_{\min} \leq u \leq u_{\max} \\ x(0) = x_{\text{initial}},\ x(T) = x_{\text{final}}.\end{aligned}$$

It would also be possible to use another cost function and further constrain the states and control input.

3.2 Discretization

To solve this numerically we need to discretize the system in some way. Using zero-order-hold sampling we can, for example, rewrite the linear system above as a discrete update system

$$x^{(k+1)} = \Phi x^{(k)} + \Gamma u^{(k)},$$

with

$$\Phi = e^{Ah}, \quad \Gamma = \int_0^h e^{At}\, dt B.$$

Here we have set the sampling time to h and introduced the nonstandard notation $x^{(k)} := x(t_k)$, which we shall use henceforth. See for example [Åström and Wittenmark, 1996] for the derivations.

We use the discrete update system on $N+1$ points in the interval $[0, T]$, equally spaced with step size h. The integral of the objective function is evaluated by numerical quadrature. This turns the original problem into its discrete counterpart

$$\min_{x,\,u} \sum_{k=0}^{N} u^{(k)} u^{(k)} \tag{3.1}$$

$$x^{(k+1)} = \Phi x^{(k)} + \Gamma u^{(k)}, \quad k = 0, \ldots, N-1 \tag{3.2}$$

$$x^{(0)} = x_{\text{initial}},\ x^{(N)} = x_{\text{final}} \tag{3.3}$$

$$u_{\min} \leq u^{(k)}, \leq u_{\max}, \quad k = 0, \ldots, N-1. \tag{3.4}$$

3.3 QP Formulation

Now let us put the discretized problem in a more convenient form. To simplify the notation let us stack the discrete states and control together

$$z = \begin{bmatrix} x^{(0)} \\ u^{(0)} \\ \vdots \\ x^{(N)} \\ u^{(N)} \end{bmatrix}.$$

We shall use this convention in later chapters. Rewrite the optimization problem (3.1) as

$$\min_z \underbrace{\begin{bmatrix} x^{(0)} \\ u^{(0)} \\ \vdots \\ x^{(N)} \\ u^{(N)} \end{bmatrix}^{\mathrm{T}}}_{z^{\mathrm{T}}} \underbrace{\begin{bmatrix} \mathbf{0} & & & & \\ & I & & & \\ & & \ddots & & \\ & & & \mathbf{0} & \\ & & & & I \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} x^{(0)} \\ u^{(0)} \\ \vdots \\ x^{(N)} \\ u^{(N)} \end{bmatrix}}_{z}.$$

subject to

$$\underbrace{\begin{bmatrix} I & & & & & & & \\ \Phi & \Gamma & -I & & & & & \\ & & & \ddots & & & & \\ & & & & \Phi & \Gamma & -I & \\ & & & & & & I & \mathbf{0} \end{bmatrix}}_{F} z = \underbrace{\begin{bmatrix} x_{\text{initial}} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ x_{\text{final}} \end{bmatrix}}_{g}, \tag{3.5}$$

and inequality constraints

$$\underbrace{\begin{bmatrix} \mathbf{0} & I & & & \\ \mathbf{0} & -I & & & \\ & & \ddots & & \\ & & & \mathbf{0} & I \\ & & & \mathbf{0} & -I \end{bmatrix}}_{C} z \leq \underbrace{\begin{bmatrix} u_{\max} \\ -u_{\min} \\ \vdots \\ u_{\max} \\ -u_{\min} \end{bmatrix}}_{d}.$$

If we put this in matrix form we get the optimization problem

$$\begin{aligned} &\min_{z} \frac{1}{2} z^{\mathrm{T}} H z \\ &Fz - g = 0 \\ &Cz - d \leq 0. \end{aligned}$$

This is now a Quadratic Program, a standard form for optimization problems (see [Jarre and Stoer, 2004]). The problem is constrained only by relations involving constant matrices. In the case for optimal control of systems governed by nonlinear ODEs and possibly nonlinear path constraints we would instead have ended up with a system of the form

$$\begin{aligned} &\min_{z \in \mathbb{R}^n} f(z) \\ &F_1(z) \leq \mathbf{0} \\ &F_2(z) = \mathbf{0} \end{aligned}$$

In the next chapter we look closer at these problems.

Discretizations

Looking back the choice of discretization seemed arbitrary. We simply chose a sampling rate and sampled the system accordingly; then assuming zero-order-hold we integrated the ODE to link the sampling points. For general systems, the task of integrating the ODE is not possible in closed form, even for zero-older-hold control inputs. Here it is natural to ask what type of discretizations we can use and expect a solution that converges to the analytic solution as the grid is refined. The pseudospectral method of discretization forms the basis of this thesis, and we shall return to this question.

4

Nonconvex Optimization

We shall here have a look at the general nonconvex nonlinear optimization problems that form the basis for our discussions in the coming chapters. The approach we follow is informal, as we first develop by expansion the optimality conditions. Nonlinear optimization is a huge subject, and we present the most important results only, based on [Jarre and Stoer, 2004]. We leave for the coming chapters to discuss numerical algorithms for finding solutions to the optimization problems.

4.1 A General Optimization Problem

We consider a general optimization problem, let us call it (**P**) for later reference:

$$\begin{aligned} &\min_{z\in\mathbb{R}^n} f(z) \\ &F_1(z) \leq \mathbf{0} \\ &F_2(z) = \mathbf{0} \end{aligned}$$

All appearing functions f, F_1 and F_2 are nonlinear, with

$$F_1(z) := \begin{bmatrix} f_1(z) \\ \vdots \\ f_p(z) \end{bmatrix}, \quad F_2(z) := \begin{bmatrix} f_{p+1}(z) \\ \vdots \\ f_m(z) \end{bmatrix},$$

and f, f_k, $k = 1, \dots, m$ are scalar-valued functions. Associated with $(\mathbf{P})$ is the *feasible set*

$$S = \{z \in \mathbb{R}^n \mid F_1(z) \leq \mathbf{0},\, F_2(z) = \mathbf{0}\}; \tag{4.1}$$

that is, the set of points in real n-dimensional vector space fulfilling the constraints. By solving $(\mathbf{P})$ we mean to find a, possibly local, minimum z^* of the objective function f, with $z^* \in S$.

We try to find the solution to $(\mathbf{P})$ numerically using an iterative algorithm. If no optimum exists this would make no sense, so we assume that such a point exists. Also we need some sort of *regularity* of the optimal point z^*. This means that the constraint set is such that we can find a solution in an incremental fashion; that is, in each iteration we can always take a small step within the feasible set.

4.2 Nonconvex Implications

Convex problems are nice, since for these local and global minima are the same. If an optimization algorithm converges to a local minimum for a convex problem it always converges to the global minimum. Conversely, for nonconvex problems we might end up in a local minimum, depending on our starting point (the initial value). Having knowledge of all the local minima, we might see that the one we have found is not very favorable; however, there is no way for the optimizer to know this. Figure 4.1 depicts the situation.

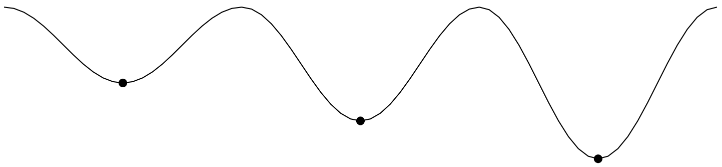

Figure 4.1 A nonconvex function with multiple minima. Depending on the starting point, a numerical optimization algorithm may end up in any of these.

4.3 An Informal Approach

Consider first an unconstrained optimization problem

$$\min_{z \in \mathbb{R}^n} f(z),$$

with the point z^* as a local minimum. Without constraints there can be no direction of descent at the minimum, that is

$$\nabla f(z^*) = \mathbf{0}. \tag{4.2}$$

Also, the Hessian at the minimum must be positive semi-definite

$$s^{\mathrm{T}} \nabla^2 f(z^*) s \geq 0, \quad s \in \mathbb{R}^n, \tag{4.3}$$

to rule out saddle points. These conditions are called the first- and second-order necessary optimality conditions for unconstrained problems.

By adding equality constraints to the unconstrained problem we get

$$\begin{gathered} \min_{z \in \mathbb{R}^n} f(z) \\ F_2(z) = \mathbf{0}. \end{gathered}$$

Starting at a local minimum z^* of the equality constrained problem we now try to take a step in a feasible direction. In each constraint this means

$$f_k(z^* + s) = f_k(z^*) + (\nabla f_k(z^*))^{\mathrm{T}} s + o(\|s\|), \quad k = p+1, \ldots, m.$$

We call the step s to remind us that it is taken to lie inside the feasible set S, even as the step size changes. This way we know that $f_k(z^* + s) = f_k(z^*) = 0$, and if we pass to the limit as s tends to zero we see that

$$(\nabla f_k(z^*))^{\mathrm{T}} s = 0, \quad k = p+1, \ldots, m,$$

for all feasible directions s. Since by assumption z^* is a minimum the objective function can not decrease in any feasible direction, hence the system

$$\begin{aligned} \nabla f(z^*)^{\mathrm{T}} s &< 0 \\ [\nabla f_{p+1}(z^*) \ \ldots \ \nabla f_m(z^*)]^{\mathrm{T}} s &= \mathbf{0} \end{aligned}$$

admits no solution. By Farka's lemma, we know that there exists a vector λ^* such that

$$\nabla f(z^*) + [\nabla f_{p+1}(z^*) \; \ldots \; \nabla f_m(z^*)]^{\mathrm{T}} \lambda^* = \mathbf{0}$$

Together with the constraints $F_2(z^*) = \mathbf{0}$, these are the so called KKT conditions of the equality constrained optimization problem. These conditions hold for optimal points in the sense that for each (local) optimal point z^* there exists a corresponding vector λ^* of *Lagrange multipliers.*

By adding the inequality constraints, $F_1(z) \leq \mathbf{0}$, we are again considering the original optimization problem (**P**). We consider very small steps only so the interesting inequality constraints are the ones that are *active*, that is $f_k(z^*) = 0$ for some $k \in [1, p]$. Expanding an active constraint using Taylor's theorem we have

$$f_k(z^* + s) = f_k(z^*) + (\nabla f_k(z^*))^{\mathrm{T}} s + o(\|s\|) = (\nabla f_k(z^*))^{\mathrm{T}} s + o(\|s\|).$$

For feasible steps s and active constraints f_k we must have $(\nabla f_k(z^*))^{\mathrm{T}} s < 0$. Again, since the objective function cannot decrease in any feasible direction at the minimum, we can write this as an unsolvable linear system

$$\begin{aligned} \nabla f(z^*)^{\mathrm{T}} s &< 0 \\ (\nabla f_k(z^*))^{\mathrm{T}} s &\leq 0, \quad k \,\text{is active} \\ [\nabla f_{p+1}(z^*) \; \ldots \; \nabla f_m(z^*)]^{\mathrm{T}} s &= \mathbf{0}. \end{aligned}$$

Once more by Farka's lemma, this gives the existence of a vector $\bar{\lambda}^* \geq \mathbf{0}$ such that

$$\nabla f(z^*) + \sum_{k \text{ active}} \bar{\lambda}_k^* \nabla f_k(z^*) + \sum_{k=p+1}^{m} \bar{\lambda}_k^* \nabla f_k(z^*) = \mathbf{0},$$

with some abuse of index notation. By augmenting $\bar{\lambda}^*$ with zeros and dropping the bar, we can instead write

$$\nabla f(z^*) + \sum_{k=1}^{m} \lambda_k^* \nabla f_k(z^*) = \mathbf{0}.$$

The constraints of λ^* together with F_1 and F_2 constitute the KKT conditions for the problem (**P**). The λ^* vector comprises the *Lagrange multipliers* associated with the optimization problem.

4.4 The KKT Conditions

The last section developed and stated the so called KKT conditions for a general nonconvex optimization problem informally. Now let us be more strict. This can be done in several ways, we base our presentation on [Jarre and Stoer, 2004]. To this end consider the optimization problem (**P**')

$$\inf\{f(z)|f_k(z) \leq 0 \text{ for } 1 \leq k \leq p, f_k(z) = 0 \text{ for } p+1 \leq m \leq p\}.$$

The theorem of Kuhn and Tucker states conditions for optimal points of this problem.

Theorem (Kuhn and Tucker) Let $C = \mathbb{R}^n$, $K = \{\lambda \in \mathbb{R}^m | y_1 \geq 0, \ldots, y_p \geq 0, y_{p+1} = \ldots = y_m = 0\}$ and the following conditions be fulfilled

- $f, f_k \in C^1(\mathbb{R}^n)$
- z^* is a local optimum of (**P**')
- (**P**') is *regular* in z^* (see 4.4)

Then there exists a $y^* \in \mathbb{R}^m$ such that

- $\lambda_k^* \geq 0$, $1 \leq k \leq p$
- $\lambda_k^* f_k(z^*) = 0$, $1 \leq k \leq m$
- $\nabla f(z^*) + \sum_{k=1}^m \lambda_k^* \nabla f_k(z^*) = \mathbf{0}$

To rule out degenerate points we need to impose some sort of regularity condition on the optimization problem. This implies that the constraint set is 'nice' in some way. Among other things it means that we can always find a direction in which to try to step when trying to solve these optimization problems numerically. We call (**P**') *regular* in $z \in S$, if 0 is an inner point of the set

$$M := F(z) + DF(z)(C - z) + K, \tag{4.4}$$

where we use

$$F(z) = [f_1(z) \ldots f_m(z)],$$

and

$$DF(z) = \begin{bmatrix} \partial_{z_1} f_1(z) & \dots & \partial_{z_n} f_1(z) \\ \vdots & \ddots & \vdots \\ \partial_{z_1} f_m(z) & \dots & \partial_{z_n} f_m(z) \end{bmatrix}.$$

Additional optimality tests, on for example second order conditions, exist but the nature of the problems make their use limited in practice. Also as the complexity of (**P**') increases determining regular points gets harder and harder. Usually these tests would not be done anyway as long as the optimization converges. Arguably most use cases are completely ignorant of issues like convergence and mathematical optimality.

5

Discretizations

We look here at ways to discretize the state-space, in which the solutions to our optimal control problems lie. This allows us to put our continuous problems in discrete form. First we talk briefly about the numerics for ODE, as this forms the basis for many simultaneous methods. We make a distinction between local and global discretizations. Local discretizations divide the interval into many small cells, each one being further divided into sub intervals. The global discretizations, often referred to as pseudospectral, use just one interval. An introduction is found in [Betts, 2001], and more recent developments can; for example, be found in [Campbell and März, 2007] and references therein.

5.1 Numerical Solution of ODEs

Consider solving the ordinary differential equation (ODE)

$$\dot{x} = f(x, t), \tag{5.1}$$

given an initial value $x(t_0) = x_0$, at some time t_0. We know that if the right hand side of the ODE (5.1) is Lipschitz continuous then the ODE has a unique solution. For numerical solution; that is, using a numerical scheme that step-wise constructs the solution, we need the scheme to be *convergent* and *consistent*. Consistency roughly means that it produces the right solution, and convergence that it does so as we decrease the numerical step size.

We speak of the numerical scheme's order as the relation between the step size and the accuracy of the solution. It is usually expressed in big-O notation.

5.2 Collocation

Collocation is the point-wise approximation of derivatives. It has given rise to several numerical schemes for solving ODE and loosely speaking the term refers to all such numerical ODE schemes. It is also used to denote all optimal control methods that work by discretizing the complete state and control input trajectories. In this section we look at collocation for solving ODEs. This is standard text book material, see for example [Deuflhard and Bornemann, 2002].

Consider solving the ODE (5.1) forward in time. Given an initial point x and stepsize h we wish to construct a polynomial u such that

$$\begin{aligned} u(t) &= x \\ u'(t + c_i h) &= f(t + c_i h,\, t + c_i h), \quad i = 1, \ldots, s; \end{aligned}$$

that is, the polynomial interpolates the point x and its derivatives are given by the function f. When designing a collocation scheme the parameters free of choice are

$$0 \leq c_1 < \ldots < c_s \leq 1,$$

with all c_i unique. The consistency/convergence limits the choice of c_i, also we wish to choose them such that the order of the numerical approximation is high.

We proceed by approximating the value of the solution at the next time step as

$$x(t + h) = u(t + h).$$

This corresponds to numerical integration. For this to work we need convergence and consistency of the scheme, as discussed in the last section. When the solution is known at the new time point, $t + h$, the process is repeated and this way we construct a point wise solution to the ODE.

If the value of the solution is sought between two grid points, the numerical solution can be repeated with a smaller time step. Remember also that the solution we have found is correct to the given accuracy, hence it does not make much sense to speak of the values in between grid points.

The collocation schemes with highest accuracies are Gauss, Radau and Lobatto. The Lobatto scheme fixes the end points, $c_1 = 0$ and $c_s = h$, and this is the scheme we use later on. Radau fixes one end point, and Gauss leaves all free to chose. The order for Gauss, Radau and Lobatto collocation, when solving ODE, are $O(h^{2s})$, $O(h^{2s-1})$ and $O(h^{2s-2})$ respectively.

5.3 Quadrature and Differentiation

Now let us grid up our optimal control problem state space using collocation from the last section. Two approaches exist — local and global. Local discretizations splits the time interval in subsections, each one then uses the a collocation scheme to fix the derivatives locally. If we only use one sub interval we end up with a global, pseudospectral, discretization.

Let us bring back our optimization problem

$$\begin{aligned} &\int_{-1}^{1} g(x,\, u)\, dt \\ &\dot{x} = f(x,\, u), \end{aligned}$$

where we have normalized the time interval to $[-1, 1]$, By affine transformations we can map any time interval $[t_0,\, t_f]$ to $[-1,\, 1]$. We discretize this problem in time with $t^{(k)}$, $k = 0, \ldots, N$, chosen as the Legendre-Gauss-Lobatto (LGL) nodes on the interval $[-1,\, 1]$. These are the points $t^{(0)} = -1$, $t^{(N)} = 1$ together with the zeros of

$$\dot{P}_N(t^{(k)}) = 0, \quad k = 1, \ldots, N-1, \tag{5.2}$$

where $P_N(t)$ is the Nth degree Legendre polynomial. It is also possible to use other discretization points, see for example [Chyba *et al.*, 2009] and [Zhang and Heping, 2008]. The Legendre polynomials solves Legendre's differential equation

$$\frac{d}{dx}\Big((1-x^2)\frac{d}{dx}P_N(x)\Big) + N(N+1)P_N(x) = 0;$$

however, the details of these polynomials are not important here. The choice of points (5.2) gives high accuracy of approximation relative to the number of discretization points when approximating the cost function integral

$$\int_{-1}^{1} g(x(t),\, u(t),\, t)\, dt \approx \sum_{k=0}^{N} w_k g(x^{(k)},\, u^{(k)},\, t^{(k)}), \tag{5.3}$$

where w_k are the quadrature weights associated with the LGL nodes. Here we have defined

$$x^{(k)} := x(t^{(k)}), \quad u^{(k)} := u(t^{(k)}), \quad k = 0, \ldots, N.$$

Henceforth we shall make use of this notation. The sum converges to the integral at *spectral rate*; that is, faster than any polynomial, as N increases.

In this thesis we use LGL nodes for our discretization needs. This rules out infinite horizon problems. To solve problems over infinite time horizons, one could, for example, use the Radau nodes, as is done in [Garg *et al.*, 2011a]. With Radau the fixed end point can be used for the initial value and the other end maps close to infinity.

We can approximate the states as

$$x(t) \approx \sum_{j=0}^{N} x^{(j)} L_j(t), \tag{5.4}$$

where $L_j(t)$ are the Lagrange polynomials of the Nth degree

$$L_j(t) = \prod_{\substack{k=0 \\ k \neq j}}^{N} \frac{t - t^{(k)}}{t^{(j)} - t^{(k)}}, \quad j = 0, \ldots, N.$$

By differentiating the states (5.4) we get an approximation of the state derivatives,

$$\dot{x}(t) \approx \sum_{j=0}^{N} x^{(j)} \dot{L}_j(t).$$

For the optimization we only need the values in the discretized points $t^{(k)}$, thus we can precompute $\dot{L}_j(t_k)$ for $j, k = 1, \dots, N$. This can be expressed as a matrix multiplication

$$\begin{bmatrix} \dot{x}_i^{(0)} \\ \vdots \\ \dot{x}_i^{(N)} \end{bmatrix} \approx D \begin{bmatrix} x_i^{(0)} \\ \vdots \\ x_i^{(N)} \end{bmatrix} \qquad i = 1, \dots, n,$$

which calculates the derivatives at the grid points along a state trajectory. The matrix elements are given by

$$D_{jk} = \begin{cases} \frac{L_N(t^{(j)})}{L_N(t^{(k)})} \frac{1}{t^{(j)} - t^{(k)}} & j \neq k \\ -\frac{N(N+1)}{4} & j = k = 0 \\ \frac{N(N+1)}{4} & j = k = N \\ 0 & \text{else} \end{cases} .$$

In the general case, with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$, and using the convention presented earlier

$$z := [x_1^{(0)} \dots x_n^{(0)}\, u_1^{(0)} \dots u_m^{(0)} \dots x_1^{(N)} \dots x_n^{(N)}\, u_1^{(N)} \dots u_m^{(N)}]^{\mathrm{T}},$$

we can simply write

$$[I_n\, 0_{n \times m}] \otimes D\, z \approx \begin{bmatrix} \frac{d}{dt} x^{(0)} \\ \vdots \\ \frac{d}{dt} x^{(N)} \end{bmatrix}$$

to calculate at once the approximation of all state derivatives in each discretization point. Here we have made use of the *Kronecker product*, which makes it easy to calculate derivatives in mathematical scripting languages. Derivation of differentiation matrices for different choices of grid points can be found in [Elgindy, 2009].

5.4 Local vs Global

Global discretization; that is, pseudospectral discretization, means using just one interval whereas local discretization uses many intervals with fewer discretization points in each. Figure 5.1 shows the difference between local and global discretizations. In each interval, be it only one or many, the same discretization technique is used, in our case we use LGL nodes. As the accuracy of the global discretization converges at spectral rate and the local as $O(h^{2s-2})$ we expect to need much less grid points for a given accuracy, and hence fewer decision variables in our discrete optimization problem this way. Less variables of course implies less computation time.

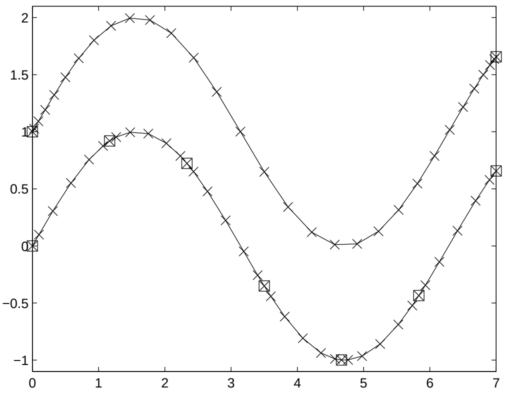

Figure 5.1 The difference between local (top) and global (bottom) discretization. The squares marks interval intersections. The points are clustered more towards the end points.

The value of the derivatives in each point depends on the values in all pseudospectral points along the trajectories. For the local discretizations, this link is not as direct as only the values at the interval intersections need to agree. As for accuracy of solution one might think that the local approach would better describe bang-bang solutions. It seems intuitive that the local discretizations capture this phenomenon better, since the interval is split up into to sub intervals and this may seem a better way to describe jumps in the control signal. However, experimental studies suggest global is better at capturing discontinuities [Huntington and Rao, 2007].

6

Pseudospectral Optimization Theory

The mathematical theory behind pseudospectral optimization has matured the last fifteen years. It has developed a link between the computer-based solution of a discrete optimization problem to its continuous counterpart. Here we give a strong enough foundation to later inspect numerical results with confidence. The results are quite straightforward but some corner cases exist. The method was first proposed in [Elnagar *et al.*, 1995]. Various versions, using different grid points exist; for example [Garg *et al.*, 2011b] and [Gong *et al.*, 2009]. See for example [Gong *et al.*, 2008] and [Kang, 2008] and references therein for a more thorough theoretic treatment.

6.1 Some Notation

We shall later be faced with two problems — the continuous problem $(\mathbf{P})$ and its discrete counterpart $(\mathbf{P}_N)$. The solution of $(\mathbf{P}_N)$ is calculated by computer but we are really interested in the solution of $(\mathbf{P})$. Naturally we would like these to agree in some sense; this is what we refer to as *convergence.*

Solutions to $(\mathbf{P})$ are in the form of state-control function pairs $t \mapsto (x(t),\, u(t))$, where the time interval is implied by the problem. Discrete solutions to $(\mathbf{P}_N)$ are of the form $(x^{(k)},\, u^{(k)})$, $k = 0, \ldots, N$. We denote optimal solutions with a star; for example, $(x^*,\, u^*)$ and $(x^{(k)*},\, u^{(k)*})$, $k =$

$0, \ldots, N$ are optimal state-control pairs for the continuous problem (**P**) and discrete problem ($\mathbf{P}_N$) respectively.

6.2 The Continuous Problem (P)

We consider here a general optimization problem governed by a set of ordinary differential equations, constrained by path inequalities and endpoint constraints. We call it (**P**) and its solution is an optimal state-control function pair $(x,\, u)$ on the normalized interval $[-1,\, 1]$.

$$\begin{aligned} \min_{x,u} \int_{-1}^{1} F(x,u)\, dt& \\ x' &= f(x,\, u) \\ e(x(-1),\, x(1)) &= \mathbf{0} \\ h(x,\, u) &\leq \mathbf{0}. \end{aligned}$$

The initial and end point conditions are captured in the more general equality constraint e. State and control trajectories are restricted by the path inequalities h.

For the problem to be well-posed we assume that all appearing functions are sufficiently smooth; that is, continuously differentiable with Lipschitz continuous gradients. Also we assume an optimal solution (x^*, u^*) to exist. The optimal state trajectory x^* is assumed to lie in the Sobolev space $W^{m,\infty}$, $m \geq 2$. This Sobolev space consists of the functions with distributional derivatives up to a given degree. The optimal control trajectory we assume to be at least continuous.

6.3 The Discrete Problem (P_N)

To discretize (**P**) we use LGL, Legendre-Gauss-Lobatto nodes, as discussed in the previous chapter. The discretized problem ($\mathbf{P}_N$) looks like

$$\min_{x,u} \sum^{N} w_k F(x^{(k)}, u^{(k)})$$

$$\|\sum_{l=0}^{N} D_{kl} x^{(k)} - f(x^{(k)}, u^{(k)})\| \leq (N-1)^{3/2-m}, \quad k = 0, \ldots, N$$

$$\|e(x^{(0)}, u^{(0)})\| \leq (N-1)^{3/2-m}$$

$$h(x^{(k)}, u^{(k)}) \leq (N-1)^{3/2-m}, \quad k = 0, \ldots, N,$$

where D is the differentiation matrix. The integral of problem ($\mathbf{P}$) is approximated by numerical quadrature and the derivatives of the state trajectory by matrix multiplication, as was shown in the previous chapter. The numerical approximation is given some slack with regards to fulfilling the constraints, instead of zero we use a 'small' value dependent on N and m, $(N-1)^{3/2-m}$ (see for example [Gong *et al.*, 2008] for details). In the limit we have that $\lim_{N\to\infty}(N-1)^{3/2-m} = 0$,

6.4 Feasibility

Suppose the problem ($\mathbf{P}$) has a solution, the next theorem tells us that so has ($\mathbf{P}_N$), if the grid is fine enough.

Theorem Given a feasible solution, $t \mapsto (x, u)$ for the continuous problem, and suppose $x \in W^{m,\infty}$ with $m \geq 2$. Then there exists M such that for any $N > M$ the discretized problem has a solution. This solution satisfies $u_k = u(t_k)$ and

$$\|x(t_k) - x^{(k)}\|_\infty \leq L(N-1)^{1-m}, \quad k = 0, \ldots, N,$$

with t_k LGL nodes and L a positive constant.

So if the continuous problem is feasible then so is the discretized problem, as long as the number of nodes is large enough.

6.5 Convergence

When solving ODEs numerically, the recommended practice is to resolve using a finer grid and then see if the successive solutions agree. The theory of numerical solution of ODEs tells us that certain methods produce the correct result, but only when the step size is small enough. For example, any oscillatory behavior should be viewed by suspicion. If it remains with fixed frequency through increasing grid refinement, we know that is not a numerical artifact. Next we introduce a way to compare solutions of pseudospectral optimal control problems.

To compare solutions of increasing accuracy (i.e. larger N), we shall introduce interpolations of solutions. Given an optimal solution of the discrete problem $(\mathbf{P}_N)$, $(x^{(k)*},\, u^{(k)*})$, $k = 0, \ldots, N$, we construct interpolation polynomials as

$$x^N(t) := \sum_{k=0}^{N} x^{(k)*}\phi_k(t), \quad u^N(t) := \sum_{k=0}^{N} u^{(k)*}\psi_k(t).$$

Here ϕ_k are the Lagrange interpolation polynomials and ψ_k any continuous function such that $\psi_k(t_j) = \delta_{jk}$, where the *Kronecker delta* is defined as

$$\delta_{jk} = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases} .$$

Corresponding to each solution of $(\mathbf{P}_N)$ there is an interpolating function $(x^N(t),\, u^N(t))$. Next we shall look at sequences of interpolating functions, which we later assume to come from solutions to $(\mathbf{P}_N)$.

Definition Consider a sequence of continuous functions $\{\rho^N(t)\}_{N=N_0}^{\infty}$ with $t \in [-1,\, 1]$. If a sub-sequence of $\{\rho^N(t)\}_{N=N_0}^{\infty}$ converges uniformly to a continuous function $\rho(t)$, we call $\rho(t)$ a *uniform accumulation point* of the sequence.

Consider solving a sequence of $(\mathbf{P_N})$ problems for $N >= M$. This generates a sequence of solutions $\{(x_k,\, u_k),\, k = 0, \ldots, N\}_{N=M}^{\infty}$. The next theorem tells us that if the sequence of solutions to $(\mathbf{P_N})$ of increasing grid refinement converges, then it converges to a solution of $(\mathbf{P})$.

Theorem Let $\{(x_k,\, u_k),\, k = 0, \ldots, N\}_{N=M}^{\infty}$ be a sequence of optimal solutions to the discretized problem $(\mathbf{P}_N)$ with interpolating sequence $\{x^N(t),\, u^N(t)\}_{N=M}^{\infty}$. Moreover, assume that the sequence has a uniform accumulation point with initial element x_0^{∞} (i.e. the initial point of the state trajectory). Let $\{(q(t),\, u^{\infty}(t))\}$ be a uniform accumulation point of the sequence $\{(x^N(t),\, u^N(t))\}$. Then $u^{\infty}(t)$ is an optimal control to the original continuous problem $(\mathbf{P})$ and

$$x^{\infty}(t) = x_0^{\infty} + \int_{-1}^{t} q(s)\, ds$$

is the corresponding optimal solution.

This means that we can run the optimizer on the discretized problem and check convergence. If we convergence, in the sense outlined above, we have converged to an optimal solution. Figure 6.1 shows a convergent control input.

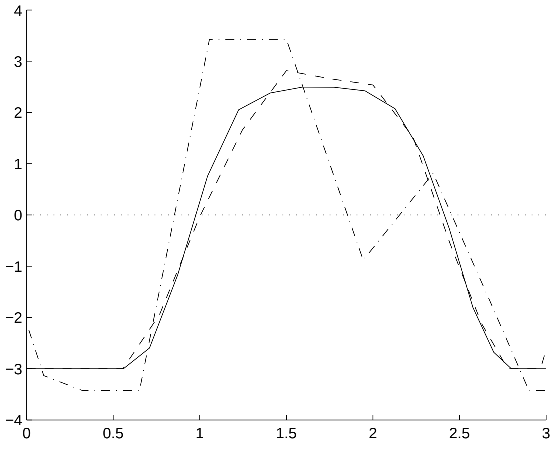

Figure 6.1 Convergence of a pseudospectral method. For coarse grids no feasible solution is found. The dash-dotted line is the solution at the coarsest grid, followed by the dashed and full line refinements. On the large grid the solution diverges and fails to approximate the function properly, but as the number of grid points is increased the solution converges.

7

Numerical Solution

Previous chapters discussed discretization of optimal control problems and conditions for optimality. If our optimization problem has an optimal solution, the theory tells us that we can discretize the problem and then try to solve the problem numerically. If the grid is fine enough then a solution will be found, given appropriate initial conditions. We discuss here different optimization algorithms for general nonconvex problems. For ease of presentation we start out by considering a linear method before moving to the more advanced nonlinear alternatives. We also show the structure of the linear system we seek to parallelize in Chapter 8.

7.1 Numerical Optimization Methods

Many numerical algorithms work iteratively trying to find a (local) optimal solution to

$$\begin{aligned} \min_{z \in \mathbb{R}^n} f(z) \\ F_1(z) \leq \mathbf{0} \\ F_2(z) = \mathbf{0} \end{aligned}$$

starting from an initial guess z_0. To this end they look for a Kuhn-Tucker pair; that is, a point (z, λ) that fulfills the KKT conditions. In each iteration they try to find a direction of descent, in which the objective function decreases, at the same time being feasible. If this is the case we step in this direction. When we reach a point where the Kuhn-Tucker conditions are fulfilled up to the tolerance of the solver, we abort and declare a minimum.

Optimization theory states optimality conditions that should hold for a point to be an optimum. Usually one does not really bother with checking optimality conditions as this is often cumbersome. Comparing this to the numerical solution of an ODE the user does not usually bother to check to see if the ODE admits a unique solution prior to simulating it. However; a solid theoretical foundation is needed, otherwise one would not know what the solution is meant to represent.

Popular methods for solving nonlinear optimization problems include inner point methods and Sequential Quadratic Programming (SQP), see any standard reference; for example, [Jarre and Stoer, 2004]. The SQP method solves a quadratic optimization problem, which is much easier than the original problem in each iteration step. Inner-point methods work on an augmented system which forces the current iterate to lie completely inside the feasible set. As the iterates proceed we get closer and closer to the boundary, given that is where the minimum is.

7.2 The Mehrotra Method

The Mehrotra method [Jarre and Stoer, 2004] is used to solve Quadratic programs of the form

$$\begin{aligned}\min_z\ &\frac{1}{2}z^{\mathrm{T}}Hz\\ &Fz-g=0\\ &Cz-d\leq 0.\end{aligned}$$

The Lagrangian for this problem is

$$L(z,\lambda_1,\lambda_2)=\frac{1}{2}z^{\mathrm{T}}Hz+c^{\mathrm{T}}z+\lambda_1^{\mathrm{T}}(Fz-g)+\lambda_2^{\mathrm{T}}(Cz-d),$$

and the corresponding KKT conditions

$$\begin{aligned} Hz^* + c + F^{\mathrm{T}}\lambda_1^* + C^{\mathrm{T}}\lambda_2^* &= 0 \\ Fz^* - g &= 0 \\ (\lambda_2^*)^{\mathrm{T}}(Cz^* - d) &= 0, \quad \lambda_2^* \geq 0 \\ Cz^* - d &<= 0. \end{aligned}$$

Now we move to solving the problem numerically. We introduce *slack* variables t as

$$Cz + t - d = 0, \quad t \geq 0,$$

and define

$$\Psi(z, \lambda_1, \lambda_2, t) = \begin{bmatrix} Hz + c + F^{\mathrm{T}}\lambda_1 + C^{\mathrm{T}}\lambda_2 \\ Fz - g \\ Cz + t - d \\ \Lambda_2 t \end{bmatrix},$$

with $\Lambda_2 = \mathrm{diag}(\lambda_2)$. We try to locate an optimum by finding a zero of Ψ iteratively by constrained Newton stepping. The Newton step becomes

$$\Psi(z + \Delta z, \lambda_1 + \Delta\lambda_1, \lambda_2 + \Delta\lambda_2, t + \Delta t) \approx \\ \Psi(z, \lambda_1, \lambda_2, t) + D\Psi(z, \lambda_1, \lambda_2, t) \begin{bmatrix} \Delta z \\ \Delta\lambda_1 \\ \Delta\lambda_2 \\ \Delta t \end{bmatrix},$$

where $D\Psi$ denotes the Jacobian of Ψ. As we iterate towards the optimimum we have to make sure that λ_2 and t are non-negative, this corresponds to staying inside the feasible set. The Newton step is determined by

$$\begin{bmatrix} H & F^{\mathrm{T}} & C^{\mathrm{T}} & \\ F & & & \\ C & & & I \\ & & T & \Lambda_2 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\lambda_1 \\ \Delta\lambda_2 \\ \Delta t \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}, \tag{7.1}$$

with $T = \operatorname{diag}(t)$, and

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = - \begin{bmatrix} Hz + c + F^{\mathrm{T}}\lambda_1 + C^{\mathrm{T}}\lambda_2 \\ Fz - g \\ Cz + t - d \\ \Lambda_2 t \end{bmatrix}.$$

Many methods involve solving systems of this kind. The Mehrotra algorithm solves (7.1) for two different right hand sides in each iteration.

Later we shall look at parallel solutions to linear systems of this kind. To this end we shall work with a symmetric reformulation, which we develop here. From the fourth row in (7.1) we have

$$\Delta t = \Lambda_2^{-1}(r_4 - T\Delta\lambda_2),$$

which we use to eliminate Δt from the system. This results in a symmetric formulation

$$\begin{bmatrix} H & F^{\mathrm{T}} & C^{\mathrm{T}} \\ F & & \\ C & & \Sigma \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta y_1 \\ \Delta y_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 - \Lambda_2^{-1} r_4 \end{bmatrix}, \tag{7.2}$$

with $\Sigma = -\Lambda_2^{-1}T$. We can work with this system and then recover Δt by

$$\Delta t = r_3 - C\Delta z,$$

in case the algorithm needs it.

7.3 Inner-point Algorithms

Now lets turn to general, not necessarily convex, optimization problems. In the last twenty-five years the inner-point methods have gained in popularity. The code IpOpt [Wächter and Biegler, 2006] is an example of an implementation. A variant is also available in MATLAB's `fmincon` solver. We give here an overview of a general inner-point optimizer. The algorithm we describe is generic, the more advanced codes build on this base.

For the general nonlinear optimization problem

$$\min_{z\in\mathbb{R}^n} f(z) \tag{7.3}$$

$$F_1(z) \leq \mathbf{0} \tag{7.4}$$

$$F_2(z) = \mathbf{0}, \tag{7.5}$$

the KKT conditions are

$$\begin{aligned}
\nabla f(z^*) + \sum_{k=1}^{m} \lambda_k \nabla f_k(z^*) &= \mathbf{0} \\
f_k(z^*)\lambda_k^* &= 0, \quad k = 1, \dots, p \\
f_k(z^*) \leq 0,\, \lambda_k^* &\geq 0, \quad k = 1, \dots, p \\
f_k(z^*) &= 0, \quad k = p+1, \dots, m
\end{aligned}$$

where λ are the Lagrange multipliers.

We try to find the optimum and the Lagrange multipliers for an augmented system looking like

$$\begin{aligned}
F_1(z) + s &= \mathbf{0}, \quad s > \mathbf{0} \\
F_2(z) &= \mathbf{0} \\
\nabla f(z) + (\lambda^{\mathrm{T}} DF(z))^{\mathrm{T}} &= \mathbf{0} \\
\Lambda s &= \mu \quad , \quad \lambda > \mathbf{0},
\end{aligned}$$

with $\Lambda = \operatorname{diag}(\lambda)$. This represents the KKT conditions of the nonlinear problem (7.3) along with slack variables. Here extra variables s and μ have been added, to keep the current iterate inside the constraints. We define $\Lambda := \operatorname{diag}(\lambda)$ and $\quad := [1 \dots 1]^{\mathrm{T}}$. Stacking all appearing variables in a vector we try to find a zero of the augmented system, a minimum of the original optimization problem, by some Newton type iteration.

As the iterations proceed the parameter μ is decreased. This allows us to move closer to the boundary of the feasible set, if that is where the optimum is located. The fact that the iterates are taken to lie within the feasible set; that is, the iterate is an inner point of the set, gives the method its name.

We group all equality relations of the augmented system together and define

$$\Psi_\mu(z,\, s,\, \lambda) := \begin{bmatrix} F_1(z) + s \\ F_2(z) \\ \nabla f(z) + (\lambda^{\mathrm{T}} DF(z))^{\mathrm{T}} \\ \Lambda s - \mu \end{bmatrix}.$$

Now we seek a zero of $\Psi_\mu(z^*,\, s^*,\, \lambda^*) = 0$ given an initial guess $(z_0^*,\, s_0^*,\, \lambda_0^*)$. An appropriate Newton type step can be derived from

$$\mathbf{0} = \Psi_\mu(z^*, s^*, \lambda^*) = \Psi_\mu(z + \Delta z, s + \Delta s, \lambda + \Delta\lambda) \approx \\ \Psi_\mu(z, s, \lambda) + D\Psi_\mu(z, s, \lambda) \begin{bmatrix} \Delta z \\ \Delta s \\ \Delta\lambda \end{bmatrix}.$$

The length of the step is then determined, assuring that the constraints on s and λ are satisfied.

On an implementational note, when using a numerical optimization code we need to implement the gradient of f and the jacobians of F_1 and F_2. It is often possible for the optimizer to approximate the derivatives numerically but this results in performance loss and accuracy loss. Highly nonlinear problem might fail to convergence in cases like this. Also sometimes the Hessian can be given to the optimizer; however, this can more easily be approximated, for example using Broyden-Fletcher-Goldfarb-Shanno (BFGS) updates [Jarre and Stoer, 2004].

Most part of the optimization time is spent solving the linear system of the Newton type step. The system is symmetric, or can made this way by clever augmentation. This way a symmetric solver can be utilized, which saves time. The appearing system is then symmetric indefinite and, for example; an LDL^{T} factorization [Duff *et al.*, 1991] can be used. If the matrix is sparse, as is often the case when solving optimal control problems using local discretization (see e.g. [Betts, 2001]), a sparse solver is recommended. For example, the solver IpOpt works solely on sparse systems [Wächter and Biegler, 2006]; however using sparse solvers to solve dense systems of course results in a performance loss.

7.4 SQP

The Sequential Quadratic Programming (SQP) algorithm also works by iteratively finding a zero of the KKT conditions [Jarre and Stoer, 2004]. It works with the original KKT system, not an augmented version which is the case for the inner point algorithm. Given an initial iterate a Newton step is taken to further get closer to an optimum. Each such step gives rise to a quadratic problem, a problem which is then solved and a step is taken. The final solution is obtained by solving a sequence of quadratic programs, this gives the method its name.

8

Parallel Solution

We turn now to parallel solution of dynamic optimization problems. Here we show that speedup can be achieved by parallel solution of the linear system that is solved in each Newton type step of the optimization algorithm. Optimization algorithms spend most of their time solving this system. We show how to proceed for a discrete update system, analogous to the example in Chapter 3, based on local discretizations and the Mehrotra method presented in Chapter 7. Trying to speed up large computations is common, see for example [Laird and Biegler, 2006] and [Tanartkit and Biegler, 1996]. Many deal with DAE since they naturally have more variables [Cervantes and Biegler, 1998].

8.1 Domain Subdivision

We refer here again to the example and discretization in Chapter 3, where we formed an optimization problem with a discrete update system as constraint

$$x^{(k+1)} = \Phi x^{(k)} + \Gamma u^{(k)},$$

and initial condition prescribed. Also the controls are interval limited

$$u_{\min} \leq u^{(k)} \leq u_{\max}.$$

Here we skip the end point constraints, as this makes the derivation less cluttered.

Previously, in chapter 7, we derived a symmetric linear system corresponding to the Mehrotra method

$$M' = \begin{bmatrix} H & F^{\mathrm{T}} & C^{\mathrm{T}} \\ F & & \\ C & & \Sigma \end{bmatrix}. \tag{8.1}$$

We look now at solving this system in parallel. For linear examples the Mehrotra method suffices; however, the principle is the same for nonlinear optimization algorithms as they too solve similar linear systems.

Let us use $2N$ sample points, as this allows us to split the interval in half easily. We write the discrete update system as

$$\begin{aligned} x^{(1)} &= \Phi x^{(0)} + \Gamma u^{(0)} \\ &\vdots \\ x^{(2N)} &= \Phi x^{(2N-1)} + \Gamma u^{(2N-1)}. \end{aligned}$$

We now introduce n extra constraints $d = x^{(N)}$, one for each state. This way we can rewrite it as

$$\begin{aligned} x^{(1)} &= \Phi x^{(0)} + \Gamma u^{(0)} \\ &\vdots \\ x^{(N)} &= \Phi x^{(N-1)} + \Gamma u^{(N-1)} \\ d &= x^{N} \\ x^{(N+1)} &= \Phi d + \Gamma u^{(N)} \\ &\vdots \\ x^{(2N)} &= \Phi x^{(2N-1)} + \Gamma u^{(2N-1)}. \end{aligned}$$

This naturally partitions the appearing variables as

$$z_1 = \begin{bmatrix} x^{(0)} \\ u^{(0)} \\ \vdots \\ u^{(N-1)} \\ x^{(N)} \end{bmatrix}, \quad z_2 = \begin{bmatrix} u^{(N)} \\ x^{(N+1)} \\ u^{(N+1)} \\ \vdots \\ x^{(2N)} \end{bmatrix}, \quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}.$$

This way the F matrix of 3.5 turns into the new extended equality constraints[1]

$$\begin{bmatrix} F_1 & & \\ & F_2 & F_2'' \\ F_1' & & -I \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{8.2}$$

The sub matrices are

$$F_1 = \begin{bmatrix} \Phi & \Gamma & -I & & & & \\ & & & \ddots & & & \\ & & & & \Phi & \Gamma & -I \end{bmatrix},$$

and

$$F_2 = \begin{bmatrix} \Gamma & -I & & & & & \\ & \Phi & \Gamma & -I & & & \\ & & & & \ddots & & \\ & & & & \Phi & \Gamma & -I \end{bmatrix}.$$

The superscripted matrices read

$$F_1' = \begin{bmatrix} 0 & \dots & 0 & I \end{bmatrix},$$

[1]For ease of presentation we have omitted any initial or final states prescribed by the optimization problem.

and

$$F_2'' = \begin{bmatrix} \Phi \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Now the linear system matrix corresponding to (8.1) can be written as

$$M = \begin{bmatrix}
H_1 & & & F_1^{\mathrm{T}} & & F_1'^{\mathrm{T}} & C_1^{\mathrm{T}} & \\
 & H_2 & & & F_2^{\mathrm{T}} & & & C_2^{\mathrm{T}} \\
 & & & & F_2''^{\mathrm{T}} & -I & & \\
F_1 & & & & & & & \\
 & F_2 & F_2'' & & & & & \\
F_1' & & -I & & & & & \\
C_1 & & & & & & \Sigma_1 & \\
 & C_2 & & & & & & \Sigma_2
\end{bmatrix}.$$

By a symmetric permutation, tantamount to what we did in Chapter 7 , we get

$$PMP^{\mathrm{T}} = \left[\begin{array}{cccccc|cc}
H_1 & F_1^{\mathrm{T}} & C_1^{\mathrm{T}} & & & & F_1'^{\mathrm{T}} & \\
F_1 & & & & & & & \\
C_1 & & \Sigma_1 & & & & & \\
 & & & H_2 & F_2^{\mathrm{T}} & C_2^{\mathrm{T}} & & \\
 & & & F_2 & & & & F_2'' \\
 & & & C_2 & & \Sigma_2 & & \\
\hline
F_1' & & & & & & & -I \\
 & & & & F_2''^{\mathrm{T}} & & -I &
\end{array}\right]. \tag{8.3}$$

Now define

$$W_k = \begin{bmatrix} H_k & F_k^{\mathrm{T}} & C_k^{\mathrm{T}} \\ F_k & & \\ C_k & & \Sigma_k \end{bmatrix},$$

and write (8.3) as

$$PMP^{\mathrm{T}} = \begin{bmatrix} W_1 & & D_1^{\mathrm{T}} \\ & W_2 & D_2^{\mathrm{T}} \\ D_1 & D_2 & T \end{bmatrix},$$

which has an arrow head form. Here D_1 and D_2 can be found by inspecting (8.3).

If we instead had worked with m sub intervals we would have ended up with

$$\begin{bmatrix} F_1 & & & & & \\ & F_2 & & & & F_2'' \\ & & \ddots & & & \vdots \\ & & & F_{N-1} & & F_{N-1}'' \\ & & & & F_N & F_N'' \\ F_1' & F_2' & \dots & F_{N-1}' & & -I \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{N-1} \\ z_N \\ d \end{bmatrix} = \begin{bmatrix} F_1 z_1 \\ F_2 z_2 + F_2'' d \\ \vdots \\ F_{N-1} z_{N-1} + F_{N-1}'' d \\ F_N z_N + F_N'' d \\ \sum_{k=1}^{N-1} F_k' z_k - d \end{bmatrix} = \begin{bmatrix} g \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The resulting symmetric arrow head system matrix is now

$$PMP^{\mathrm{T}} = \begin{bmatrix} W_1 & & & D_1^{\mathrm{T}} \\ & \ddots & & \vdots \\ & & W_m & D_m^{\mathrm{T}} \\ D_1 & \dots & D_m & T \end{bmatrix}, \tag{8.4}$$

where for each k the corresponding D_k would be a sparse matrices made up of zeros, F_k' and F_k''. The D_k matrices are long and thin when the number of data points in each interval N is large.

8.2 Parallel Solution

Consider solving the arrow head system derived in the previous section

$$\begin{bmatrix} W_1 & & & D_1^{\mathrm{T}} \\ & \ddots & & \vdots \\ & & W_N & D_N^{\mathrm{T}} \\ D_1 & \dots & D_N & T \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ w_T \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \\ b_T \end{bmatrix}, \tag{8.5}$$

where we here use an arbitrary right hand side consisting of b_k. Rewrite (8.5) as

$$\begin{cases} W_1 w_k + D_1^{\mathrm{T}} w_T & = & b_1 \\ & \vdots & \\ W_N w_k + D_N^{\mathrm{T}} w_T & = & b_N \\ \sum_{k=1}^{N} D_k w_k + T w_T & = & b_T \end{cases}$$

Formally we have

$$w_k = W_k^{-1}(b_k - D_k^{\mathrm{T}} w_T), \quad k = 1, , \dots, N,$$

and from this we get

$$(T - \sum_{k=1}^{N} D_k W_k^{-1} D_k^{\mathrm{T}}) w_T = b_T - \sum_{k=1}^{N} D_k W_k^{-1} b_k. \tag{8.6}$$

Here the Schur complement

$$S = T - \sum_{k=1}^{N} D_k W_k^{-1} D_k^{\mathrm{T}}$$

is dense.

If we start by factoring W_k, we can then solve

$$W_k z_k = b_k, \quad k = 1, \dots, N,$$

and

$$W_k Z_k = D_k^{\mathrm{T}}, \quad k = 1, \dots, N,$$

for z_k and Z_k. This can be done in parallel. Now (8.6) turns into

$$(T - \sum_{k=1}^{N} D_k Z_k) w_T = b_T - \sum_{k=1}^{N} D_k z_k$$

which we solve for w_T. This must be done on a single CPU, and must be completed before we can proceed.

Next solve

$$W_k w_k = b_k - D^{\mathrm{T}} w_T, \quad k = 1, \ldots, N,$$

for w_k, again in parallel.

For the pseudospectral case we would instead of using only one differentiation matrix we would make use of one matrix for each subinterval.

8.3 Numbers

We test the proposed algorithm on a current PC @ 2.66 GHz running Linux, using `FORTRAN95` for the implementation. We make use of randomly chosen linear systems; that is, with no real world connection, for the tests. With three states gives the following results

$N \setminus$ CPUs	1	2	4
32	0.062	0.069	0.074
64	0.12	0.10	0.14
128	0.49	0.35	0.41
256	5.38	2.81	1.52
512	47.1	24.1	12.1

The timings are in seconds and measures the computation time per iteration. Since most of the time of the optimization is spent solving the linear system, the one that we solve on multiple cores, a considerable speed up can be increased. For the example with few grid points; however, we loose total time because of the overhead induced by the parallelization.

8.4 Conclusion

As seen by the numerical experiments in the last section for small systems the overhead takes considerable time, and as such the parallelization scheme proposed results in longer computation times. For larger systems; however, a considerable speed-up can be achieved as long as the number of grid points N increases with compared to the number of states n. But, in real applications usually not that many discretization points are needed to get a good enough accuracy, and as such the proposed method is not of much use for the current application.

Others have shown similar possibilities in speed up [Laird *et al.*, 2011], also based on randomized systems. The class of systems here is enlarged to include Differential Algebraic Equations (DAE). As the optimization of DAEs lacks theoretical foundation in literature running optimizations on random systems is questionable. In the case of the proposed parallelization algorithm the extra number of algebraics increases the size of the linear system without increasing the coupling (as described by d in the matrix of (8.2)). However, as long as systems with industrial relevance stay elusive, both for linear ODE and DAE systems, these theoretical speed-ups are of little use. In theory the time spent on solving the linear system can be divided by the number of available CPUs. In the limit the speed-up factor equals the number of available processors.

9 Pseudospectral Model Predictive Control

Here we present pseudospectral control in an Model Predictive Control (MPC) setting, see [Magni *et al.*, 2009] and [Rao *et al.*, 1998]. When solving an optimal control problem over a given time interval we are really doing open-loop control, this is not desirable. In a feedback setting the optimizer would run many times, using a receding horizon, and the result of the previous optimization is used for controlling the plant until a new control is calculated by the optimizer. The computation time depends on the size of the problem and thus on the discretization of the model. With long computation times we need an accurate model so that the system does not drift away. Pseudospectral optimization enables low-dimensional numerical optimization problems, due to the high accuracy of approximation. This cuts computation time severely, in comparison to other methods [Huntington and Rao, 2007].

9.1 Pseudospectral Optimization in Automatic Control

The use of pseudospectral collocation methods initially found widespread use in the solution of partial differential equations (PDE), as an alternative to finite differences and finite elements. Presently they can be considered a well developed technology, see for example [Fornbern, 1998], They were first applied to the field of automatic control in [Elnagar *et al.*, 1995]. So far they have not received a thorough treatment by the control community, although [Gong *et al.*, 2007] is a step in this direction. Examples are scarce, but include [Song and Dyke, 2011].

9.2 MPC Controller Structure

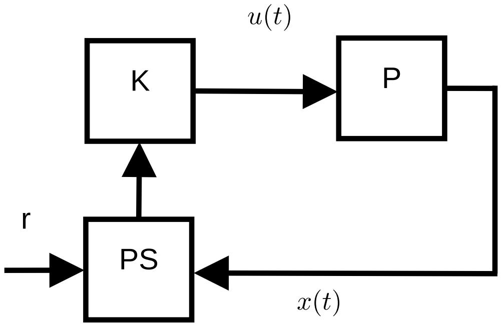

Figure 9.1 Pseudospectral MPC controller structure. The actuator K uses the last computed results to generate and input for the plant P. Computations are carried out int the pseudospectral block PS.

The controller structure is shown in Figure 9.1. The output of the plant, P, is fed to the pseudospectral optimizer, PS. After computing a control trajectory PS hands it to the actuator K which at a given sampling rate gives inputs to the plant. When the optimizer is done with the computations it once again samples the plant and starts a new computation and the cycle repeats.

The plant P in this case is a system that can be described by an ODE

$$\dot{x} = f(x, u).$$

The optimizer PS implements a pseudospectral optimal control algorithm as described in previous chapters.

The actuator K is run at a fixed sampling rate, at a higher frequency than the optimizer PS. At the sampling points of K the value of the control signal is computed using interpolation

$$u(t) = \sum_{k=0}^{N} u_k L_k(t),$$

where u_k is the control sequence calculated by the optimizer. We could also use a resampled version of the control sequence or parts of it since the optimization horizon will be equal to or further off in the future than the next expected control update.

Note that at the start of a control sequence there is no precomputed control signal. We assume that we start off at an equilibrium, produced e.g. by a constant control signal. It would also be possible to stabilize the plant using a simpler controller, e.g. a PID-controller, but then K must also be fed the output of the plant.

9.3 Optimization Loop

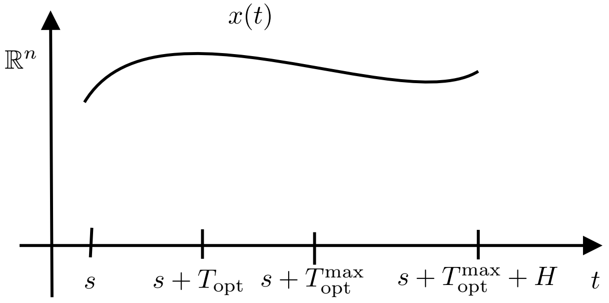

Figure 9.2 Control loop timing.

The timing of the optimization loop is shown in Figure 9.2. Here T_{opt} is the computation time of the current optimization, $T_{\text{opt}}^{\max}$ is the maximum allowed computation time and H is the time horizon of the optimization problem.

1. The plant P is sampled at time s, which gives $x(s)$.
2. Using $x(s)$ as starting point we simulate the plant

$$x(s + T_{\text{opt}}^{\max}) = x(s) + \int_s^{s+T_{\text{opt}}^{\max}} f(x(t),\, u(t))\, dt,$$

   using some numerical ODE method. This way we calculate the state of the system, given the current control sequence, at the time the optimization is expected to have finished.

3. An optimal control is then solved using $x(t + T_{\text{opt}}^{\max})$ as initial value
$$x^0 = x(s + T_{\text{opt}}^{\max}),$$
over the horizon H. This corresponds to $t_0 = s + T_{\text{opt}}^{\max}$ and $t_f = s + T_{\text{opt}}^{\max} + H$.
4. The control sequence given at the nodes of the optimization are fed to the actuator K at time $s + T_{\text{opt}}^{\max}$. If the actual computation time exceeds $t_{\text{opt}}^{\max}$ we stop and hand the current iterate to the actuator.
5. Return to the first step and iterate.

We need an estimate for the computation time to use as $T_{\text{opt}}^{\max}$. This can be acquired from simulation runs. If the optimizations need to be stopped prematurely in step four of the last section then we might need to increase our computation time estimate. It would be possible to use some sort of adaptive scheme to estimate the computation time as the iterations proceed. In the open-loop scenario many optimization runs may be needed to determine a suitable grid refinement. At coarser grids the problem might not converge. Suitable parameters can be found by repeated simulations.

9.4 Numerical Evaluation

We now proceed to test the proposed pseudospectral controller on a standard control problem, a nonlinear double tank.

Double Tank Control

Consider the double tank process in Figure 9.3. It comprises two tanks and a pump inlet to the upper tank. The states are the liquid levels of the tanks, x_1 and x_2. Liquid is fed to the upper tank using a pump, which acts as our control signal, u. Assuming laminar flow the dynamics can be written
$$\begin{cases} x_1' = -a_1\sqrt{x}_1 + bu \\ x_2' = a_1\sqrt{x}_1 - a_2\sqrt{x}_2 \end{cases},$$

The constant b depends on the pump, which we assume to be linear. The amount of liquid that leave the tanks is proportional to the square root

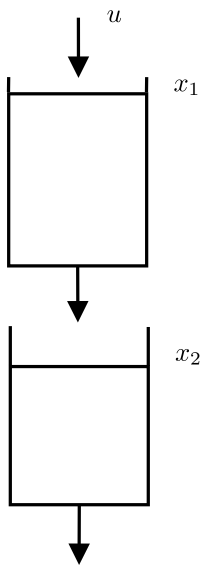

Figure 9.3 Two interconnected tanks with one inlet.

of the liquid level, according to Torricelli's law. Also the density, tank outlet and geometry effect the outflow. This is captured in the constants a_k, $k = 1, 2$.

The aim here is to control the water level of the lower tank, x_2, to a new set point. This puts constraints on the states, as the tanks have limited volume with a non-negative water level. The actuator also limits the feasible controls since the pump has limited range and cannot suck water from the upper tank; that is, the control input is positive.

Optimization Problem

The optimal control problem is to minimize

$$\min_{x,u} \int_{t_0}^{t_f} (x_2(t) - x_2^{\text{ref}}(t))^2 \, dt$$

subject to the dynamics

$$\begin{cases} x_1' = -a_1\sqrt{x}_1 + bu \\ x_2' = a_1\sqrt{x}_1 - a_2\sqrt{x}_2 \end{cases},$$

with initial condition

$$x(t_0) = x^0,$$

and path inequalities

$$\left.\begin{array}{l} \mathbf{0} \le \mathbf{x(t)} \le \mathbf{x}^{\max} \\ 0 \le u(t) \le u^{\max} \end{array}\right\} \quad t \in [t_0,\, t_f].$$

The reference trajectory x_2^{ref} describes the desired water level of the lower tank as a function of time.

Simulation

We test the controller on the double tank process described earlier. In the simulation the plant is to be controlled to a new set point, and we use

$$x_2^{\text{ref}}(t) \equiv x_2^{\text{setpoint}},$$

to have the optimization penalise deviations from it. We used a value of $N = 25$ when discretizing the problem, which we found to give a fine enough grid. For the sake of the simulation we set the constants as

parameter	x_1^0	x_2^0	$x_1^{\max}$	$x_2^{\max}$	x_2^{ref}	$u^{\max}$	a_1	a_2	b
value	0.25	0.3	0.8	0.8	0.4	3	1	1	1

For these values we found a suitable maximum time for the computations as $T_{\text{opt}}^{\max} = 0.05$ by simulation. We set the time horizon to $H = 2T_{\text{opt}}^{\max}$.

The control of the system is shown in Figure 9.4. It is bang-bang type solution, which starts to fill the upper tank by giving full pump input. After a while it stops and the water in the upper tank continues to fill the lower. Later it reaches the desired set point, and pump is started to achieve a stationary tank level.

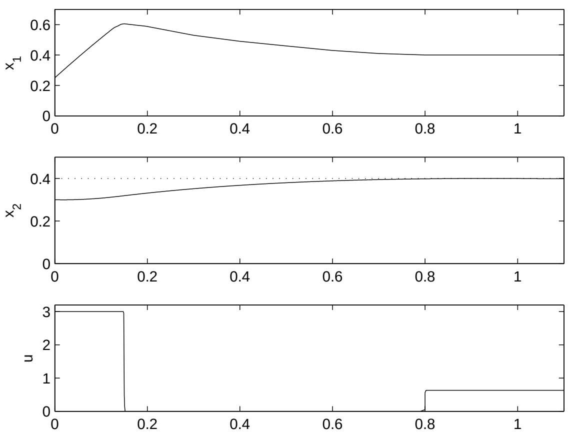

Figure 9.4 Simulation run showing the tank levels and control signal.

The simulations were carried out using the MATLAB nonlinear programming solver *fmincon* using compiled mex-files for function evaluations, on a recent desktop computer. The computation time depends on the size of problem, i.e. the discretization. The mean computational times for the optimization run were

Discretization (nodes)	CPU time (s)
PS (25)	0.027
Local (150)	3.2

In parenthesis are the number of nodes, including all sub intervals. The number of nodes were chosen as to achieve the same accuracy level. We have included the corresponding local discretization method for comparison; that is, using the same node placement but with several sub intervals. In general, local discretizations are much more time consuming than global. Further comparisons between local and global discretizations can be found in [Huntington and Rao, 2007].

9.5 Conclusions

We have proposed a pseudospectral model predictive control technique. The relatively low computation time allows us to use pseudospectral optimization for online optimization. Through numerical experiments we have shown its application to a double tank process.

Looking at the figures it looks decent; however, it took some of tuning the model and the initial conditions to avoid divergence. Even though the repeated simulations are initialized with the value of the last optimal value, and as such we expect the new value to move very little, the algorithm sometimes diverges. Even if the optimizations diverges only once, using these kinds of optimizations in real life scenarios can be disadvantageous. It would then require operator supervision, or some other type of fail safe mechanism. This suggests the plant only to be run through regions of high accuracy and nice convergence properties, presumably found by simulation, and is alluded to in[Kang and Bedrossian, 2007]. Also the need to set a maximum computation time can turn out to be problematic, if this time is exceeded[1].

In a real setting model inaccuracy will make the system drift in the simulation stage of the controller which will effect performance. If the computations are fast enough in comparison to this drift the method is appropriate. We should do trajectory changes through regions were the model is known to be good and optimizations converge, which can be verified by simulations first. To avoid drift over time it would be possible to use a different controller at the set points, and then have the proposed pseudospectral controller handle set point changes in an optimal fashion.

[1]Issues like this are all too often overlooked in the literature.

10 Pseudospectral Observers

Now we look at using pseudospectral optimization to construct observers for general nonlinear systems. We build upon the pseudospectral optimization theory developed in previous chapters. Next we look at a possible observer implementation by Qong and associates [Gong *et al.*, 2007], and we discuss its shortcomings. We then propose a fix for this, building upon the results out-of-sequence measurements and sensor fusion [Bar-Shalom, 2002]. This is proposed future work and as such we allow ourselves latitude as to detail.

10.1 An Output System

When observing the output of a system the control input is assumed to be known. This makes sense since the observer is part of the controller. This allows us to rewrite the ODE governed system dynamics

$$\dot{x} = f(x(t),\, u(t)) = \hat{f}(x(t),\, t),$$

as we henceforth treat the control input as time-dependent dynamics.

The output system we consider is now

$$\begin{aligned} \dot{x} &= \hat{f}(x(t),\, t) \\ y &= h(x(t)), \end{aligned}$$

where we may sample the output y at will or at some given rate. We may also incorporate a constraint set

$$r(x(t)) \leq \mathbf{0}.$$

This way me may add extra information of what we know about the states already; for example, that they lie within a certain interval.

In practice the sensors and microcontroller of the implementation platform limits the possible sampling rate. We shall assume that we are always allowed to use the values we present with mathematics.

10.2 Observability

For a linear system

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx,\end{aligned}$$

checking if the system is observable or not comes down to checking the rank of the observability gramian

$$W_o := \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}.$$

If the determinant of this matrix has full rank we may reconstruct the state trajectory x only by looking at the output y. In practice this means that we can not have two different state trajectories $x(t)$ that result in the same system output $y(t)$ over the time interval we observe. This makes appropriate the following observability condition for our nonlinear system:

Definition There exists a positive constant T_h such that if two state trajectories, $u(t)$ and $v(t)$, produces the same output over any interval

$$\int_{T-T_h}^{T} \|h(u(t)) - h(v(t))\|^2 \, dt = 0,$$

with $T \in [T_h, \infty]$; , then $u(t)$ must equal $v(t)$ in the interval.

We may restrict the allowed trajectories in the definition to lie within the constraint set $r(x(t)) \leq \mathbf{0}$. This way me might turn some unobservable systems into observable ones, in terms of our definition. For example, if the system under consideration is linear, time-varying and is uniformly observable, then our observability condition holds.

10.3 Optimization Problem

Assuming we have an output system satisfying the observability condition, we now phrase the observability problem as an optimization problem. Consider solving

$$\begin{aligned}\min_{\hat{x}} \int_{T}^{T-T_h} & \|h(\hat{x}(t)) - y(t)\|^2 \, dt \\ \dot{\hat{x}} &= \hat{f}(\hat{x}(t), t) \\ r(\hat{x}(t)) &\leq \mathbf{0}.\end{aligned}$$

Here we let the observed states wear a hat[1]. This way the distinction between the true and observed states and control inputs becomes clear.

The discrete counterpart becomes

$$\begin{aligned}\min_{\hat{x}} \sum_{k=0}^{N} & \|h(\hat{x}^{(k)}) - y^{(k)}\|^2 \\ \|\sum_{l=0}^{N} D_{kl} x^{(l)} - \hat{f}(\hat{x}^{(k)}, t^{(k)})\| &\leq (N-1)^{\frac{3}{2}-m}, \quad k = 0, \dots, N \\ r(\hat{x}^{(k)}) &\leq (N-1)^{\frac{3}{2}-m}, \quad k = 0, \dots, N,\end{aligned}$$

where D is the pseudospectral differentiation matrix. Here, as previously, the discretized problem needs some slack when trying to fulfill the equalities, for the optimization to converge.

To see that the problem has a solution we argue like this. The cost function is nice and smooth, given that the output function $y = h(x(t))$ is,

[1]In this formulation we include the control inputs. It would also be possible to disguise the control inputs as a time dependence.

and as such the quadratic cost should admit a solution. If the continuous problem has a solution then so has the discrete problem, if the grid is fine enough. Also since no two different state trajectories can produce the same output, assuming our definition of observability , then if the optimization converges we have recovered the states we wish to observe.

10.4 A Possible Observer Structure

We introduce here a possible observer structure, due to Gong and associates [Gong *et al.*, 2007]. The algorithm is naturally divided into two phases. In an initial phase the observer collects data by sampling the system at regular intervals, this is to collect data over a given time horizon. Next the system is sampled continuously and for each new point the optimizer is run. From the previously collected samples we can then interpolate to get the values we are looking for; that is, the values at the pseudospectral discretization points.

The initial phase proceeds as follows.

1. Select tuning parameters N and L, and initial guess $x(t_0)$. The parameter N is the number of grid points used by the optimizer, and L is the number of samples we collect for the time horizon.
2. Calculate the pseudospectral nodes and other data needed by the pseudospectral algorithm that can be done offline; for example, the differentiation matrix.
3. Simulate the system using the initial guess. This forms the initial guess of the optimization.
4. Collect initial data; that is, L equally spaced samples.

The pseudospectral algorithm needs outputs sampled at the Lobatto nodes but sampling is usually performed at equally spaced intervals, To overcome this spline interpolation is used. After the initial phase the observer repeatedly samples the system and performs a pseudospectral optimization run. The main phase of the algorithm looks like this.

1. Collect the next measurement.
2. Construct spline function to get data at the pseudospectral nodes.

3. Solve the resulting discrete optimization problem. The result of the optimization at the final time is an estimate of the system states at the current sampling time.
4. Simulate to get the system state at the next sampling point and then repeat, this will be the initial guess of the next optimization.

Let us try the proposed algorithm on our water tank example

$$\begin{cases} \dot{x}_1 = -a_1\sqrt{x}_1 + bu \\ \dot{x}_2' = a_1\sqrt{x}_1 - a_2\sqrt{x}_2 \end{cases} .$$

as introduced in Chapter 9. Using only one sensor we measure now only the water level in the lower tank; that is, setting

$$y = x_2.$$

Our aim is to measure the water level of the upper tank, x_1, by watching y.

For the experiment we use $L = 8$; that is, we use six samples to initially collect process output. The pseudospectral optimizer makes use of $N = 35$ grid points, and as proposed we use the sample values to interpolate the values at the grid. We use a constant pump input, $u = u_{\text{const}}$. The results are compared to a standard linear observer, based on linearization around the steady state corresponding to the given input and zero order hold. The results are shown in Figure 10.1, where we have not taken into account the computational delay associated with the pseudospectral optimization.

The observed states, stemming from the pseudospectral controller corresponds well to the simulated states. The linear observer converges more slowly to the true states, which approach the linearization point. In Figure 10.2 the output with optimization delay is shown. We see that the computation times vary between computations. The random compution times will be hard to account for if we wish to close the loop and use the observed states when calculating a control input.

10.5 Sensor Fusion and OOSM

A problem of the proposed observer scheme is that if we wish to close the control loop we will have problems if the computation time exceeds the

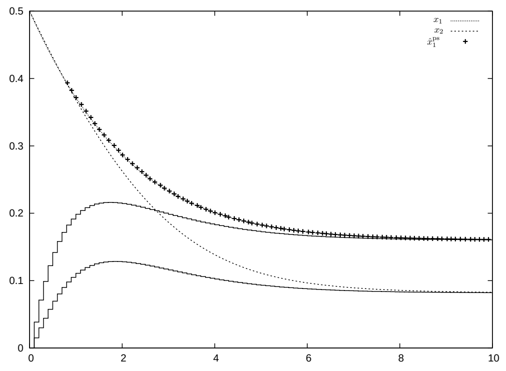

Figure 10.1 The pseudospectral observer compared to a standard linear observer. The computational delay of the optimization is not accounted for.

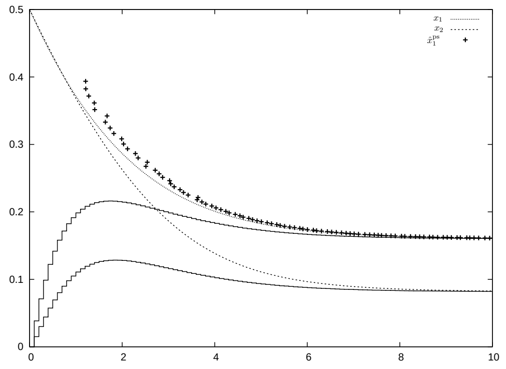

Figure 10.2 The pseudospectral observer compared to a standard linear observer, with computational delay.

sampling rate; that is, the problem must be solved online within a given time. The algorithm uses fixed sampling, this means we need to adjust the sampling rate to some worst-case computation time. If this is exceeded some fail-safe mechanism would have to be implemented. We would rather like to use the computed value as soon as possible.

Consider a system comprising a single sensor. Using the pseudospectral observer in this case we would have to rely on low model error to be successful. This is because the pseudospectral computation time is quite large compared to common sampling rates of industrial controllers, which means that for long times the controller would need to rely on the correctness of the model. A better idea would be to combine the pseudospectral observer with a fast sensor. This would mean incorporating the pseudospectral observer computations as Out-Of-Sequence Measurements (OOSM); see for example [Bar-Shalom, 2002].

In this settings the pseudospectral observer can be viewed as a, in some sense, slow sensor with random delay. This is comparable to the graphics processing used in[Berntorp *et al.*, 2011], where a computationally intensive computer vision algorithm is used in conjunction with a much faster sensor. However, in this setting we also need a faster and more steady sensor to be used in conjunction with the pseudospectral observer. Perhaps even a standard linear observer could be used as a faster sensor. In such a setting the states of the linear observer could be used for feedback, which would then periodically be corrected by using the states calculated by the pseudospectral observer. In the future we are looking to implement and test this use of pseudospectral observers more thoroughly.

11 Conclusions

This thesis has covered applications of pseudospectral optimization applied in the field of automatic control. Pseudospectral optimal control in an model predictive control (MPC) setting has been evaluated. We also propose a way to use pseudospectral optimization as a nonlinear state estimator together with Out-Of-Sequence-Measurements (OOSM). It has also dealt with parallel soluation of optimization problems governed by system dynamics described by ODE. To carry out numerical tests a programming framework has been developed, although the implementation is not the subject of the thesis.

11.1 Parallel Solution

When solving numerically optimization problems governed by ODE, much time is spent solving a linear system. We proposed a way to carry out the computations in parallel. By the numerical experiments we found the overhead related to the parallelization to take considerable time. Sometimes the parallelization scheme proposed results in longer computation times, compared to using a single CPU. For larger systems; however, a considerable speed-up can be achieved as long as the number of grid points increases. In real applications usually not that many discretization points are needed to get a good enough accuracy, and as such the proposed method can be questioned.

MPC Application

We proposed a pseudospectral model predictive control (MPC) technique. The relatively low computation time allows make possible the use of a current PC workstation for the implementation. Through numerical experiments we showed its application to a double tank process.

The results were not completely satisfying, as it took quite a lot of tuning of both the model and the initial conditions to avoid divergence. Sometimes the algorithm diverge, which makes its industrial usage limited, as it would then require operator supervision, or some other type of fail safe mechanism. This suggests the plant only to be run through regions of high accuracy and nice convergence properties, presumably found by simulation.

In a real setting model inaccuracy will make the system drift in the simulation stage of the controller which will effect performance. If the computations are fast enough in comparison to this drift the method is appropriate. We should do trajectory changes through regions were the model is known to be good and optimizations converge, if we are to use the method at all.

11.2 Observer

We looked at using pseudospectral optimization to construct observers for general nonlinear systems. The pseudospectral observer show promising results. Compared to a linear observer the observed states much better agrees with the true solution. However, the larger computaional overhead needs to be accounted for. For future work we proposed a way to incorporate a pseudospectral optimizer in the framework of Out-Of-Sequence-Measurements (OOSM), to account for the random delays inherit in the optimization based approach.

A

A Pseudospectral Optimizer MATLAB Implementation

We discuss here the implementation of a pseudospectral optimizer, using MATLAB [MATLAB, 2010] as a base. The proposed implementation utilizes the `fmincon` function for optimization, and makes use of the symbolic toolbox for automatic generation of optimization code for a `mex`-based back end. It is similar in design to the JModelica.org [JModelica.org, 2012] compiler, although much simpler.

A.1 The `fmincon` Function

The MATLAB programming package has many possibilities for optimization. To solve our nonlinear problems we use the function `fmincon`. It finds a constrained minimum of a function of several variables and has many options and variations in usage.

The optimization problem `fmincon` solves is

$$\begin{aligned} \min_{z\in\mathbb{R}^n} f(z) \\ Az \leq b \\ A_{\mathrm{eq}}z = b_{\mathrm{eq}} \\ c(z) \leq 0 \\ c_{\mathrm{eq}}(z) = 0 \\ b_{\mathrm{l}} \leq z \leq b_{\mathrm{u}}. \end{aligned}$$

Here the linear constraints are stated independently, to allow the solver to cut some corners of efficiency. MATLAB also makes special the case of linear bounds. In terms of the general nonlinear optimization problem of Chapter 4. we would have

$$F_1(z) := \begin{bmatrix} A_{\mathrm{eq}}z - b_{\mathrm{eq}} \\ c_{\mathrm{eq}}(z) \\ b_{\mathrm{l}} - z \\ z - b_{\mathrm{u}} \end{bmatrix}, \quad F_2(z) := \begin{bmatrix} Az - b \\ c(z) \end{bmatrix}.$$

A standard call would look like

```
z = fmincon(@f, z0, A,b, Aeq,beq, lb,ub, @nonlincon)
```

where `z0` is the initial and `nonlincon` a function returning the nonlinear constraints, $c(z)$ and $c_{\mathrm{eq}}(z)$.

Linear constraints are specified directly in the call by matrices. Standard or sparse matrices describe `A`, `b`, `Aeq` and `beq`.

The objective function looks like

```
function [f, dfdz] = objective(z)
    f = ...;

    if nargout > 2
        dfdz = ...;
    end
end
```

and the constraint function like

```
function [c, ceq, dcdz, dceqdz] = nonlincon(z)
    c = ...;
    ceq = ...;

    if nargout > 2
        dcdz = ...;
        dceqdz = ...;
    end
end
```

It is possible to use analytic derivatives, these are returned in the variable `dfdz` for the objective, and in `dcdz` and `dceqd` for the constraints. To turn on the functionality, use for the constraints

```
options = optimset(optimset, 'GradConstr', 'on');
```

and for the objective

```
options = optimset(optimset, 'GradObj', 'on');
```

If sparse matrices are used the jacobians of `nonlincon` must also use the sparse format.

The `fmincon` optimizer can choose from different optimization algorithms. For example, to select the solver to use an interior point solver use

```
options = optimset(optimset, 'Algorithm', 'interior-point');
```

or for an active set solver

```
options = optimset(optimset, 'Algorithm', 'active-set');
```

A.2 The `mex`-backend

The objective and constraint functions of last section, written in MATLAB scripting language, dispatch calls to underlying `mex`-functions. The main work is done in precompiled functions, as we use `mex`-files written in the C programming language to facilitate fast computations. The idea is to exploit that the discretizations of the ODEs describing the system dynamics consists of the same function evaluated in each discretization point. For example, assume that the optimization problem is governed by dynamics

$$\dot{x} = f(x(t), u(t)).$$

Using discretized variables

$$z = \begin{bmatrix} x_0 \\ u_0 \\ \vdots \\ x_N \\ u_N \end{bmatrix}.$$

we require the ODE to hold (approximately) in each discretized point as

$$Dz = \begin{bmatrix} f(x_0, u_0) \\ \dots \\ f(x_N, u_N) \end{bmatrix} \tag{A.1}$$

The differentiation matrix is D and the calculation of derivatives is most easily done by matrix multiplication directly in the MATLAB scripting language. The right hand side of (A.1) is the reevaluation of the ODE-function for different points and is programmed into a C-language `mex`-file. The grid size; that is, the number of discretization points is changed as this C-style-pseudo-code shows for a scalar ODE

```
int N = ...;
double F[N];
for (int k = 0; k <= N; k++) {
    F[k] = f(x[k], u[k]);
}
```

This allows for optimization at different accuracies, to test for convergence.

A.3 Code Generation

The MATLAB package facilitates calculation of analytic derivatives through the symbolic toolbox. We make use of this to generate first and second order functions for the solver. Consider the simple ODE

$$\begin{cases} \dot{x}_1 = x_1^2 \\ \dot{x}_2 = 2x_1 + x_2 \end{cases}.$$

We need to calculate the jacobian of the right hand side to be able to generate second order information for the solver. The calculation of the jacobian is easily done using the symbolic toolbox as

```
syms x1 x2
f = [x_1*x_1; 2*x1 + x2];
variables = [x1, x2];
J = jacobian(f, variables)
```

This is used in each grid point to form a large matrix, typically looking like

$$\begin{bmatrix} J_0 & & \\ & \ddots & \\ & & J_N \end{bmatrix},$$

where we make use of J_k to mean the jacobian evaluated at point (x_k, u_k).

A.4 Example Usage

The top level interface of our MATLAB optimizer specifies the optimization functions as strings. By some clever use of the `sprintf` and `eval` functions we can make use ot the symbolic toolbox for automatic code generation. Various objective functions can be used, the quadratic is common and very easy to implement by squaring the appearing variables and multiplying by the pseudospectral quadrature weights.

An example problem formulation would look like

```
ode_rhs = {'x2', 'x3^3', 'u'};
vars = {'x1', 'x2', 'x3', 'u'};

time_final = 3.0;
N = 32;
```

Input constraints can then be specified as

```
b1 = 3.0*ones(pee.N+1, 1);
A1 = kron(eye(pee.N+1), [0 0 0 1]);

A = [A1; -A1];
b = [b1; b1];
```

to limit the control input to $|u| \leq 3$.

After the generation of nonlinear constraints and objective function we can call the `fmincon` to finish the optimization.

B

Bibliography

Arnold, V. I. (1978): *Mathematical methods of classical mechanics.* Springer.

Bar-Shalom, Y. (2002): "Update with out-of-sequence measurements in tracking: Exact solution." *IEEE Transactions on Aerospace and Electronic Systems*, **38:3**, pp. 769–778.

Berntorp, K., K.-E. Årzén, and A. Robertsson (2011): "Sensor fusion for motion estimation of mobile robots with compensation for out-of-sequence measurements." In *Proc. 11th international conference on control, automation and systems.*

Betts, J. T. (2001): *Practical methods for optimal control using nonlinear programming*, vol. 3 of *Advances in Design and Control.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Betts, J. T., N. Biehn, and S. L. Campbell (2002): "Convergence of nonconvergent irk discretizations of optimal control problems with state inequality constraints." *SIAM journal of scientific computing*, **23:6**, pp. 1981–2007.

Campbell, S. L. and R. März (2007): "Direct transcription solution of high index optimal control problems and regular euler-lagrange equations." *Journal of computational and applied mathematics*, **202**, pp. 186–202.

Cervantes, A. and L. T. Biegler (1998): "Large-scale DAE optimization using a simultaneous nlp formulation." *AIChe Journal*, **44:5**, pp. 1038–1050.

Chyba, M., E. Hairer, and G. Vilmart (2009): “The role of symplectic integrators in optimal control.” *Optimal control applications and methods*, **30**, pp. 367–382.

Deuflhard, P. and F. Bornemann (2002): *Numerische Mathematik II*. Walter de Gruyter.

Duff, I. S., N. I. M. Gould, J. K. Reid, and J. A. Scott (1991): “The factorization of sparse symmetric indefinite matrices.” *IMA journal of numerical analysis*, **11**, pp. 181–204.

Elgindy, K. T. (2009): “Generation of higher order pseudospectral integration matrices.” *Applied Mathematics and Computation*, **209:2**, pp. 153–161.

Elnagar, G., M. A. Kazemi, and M. Razzaghi (1995): “The pseudospectral legendre method for discretizing optimal control problems.” *IEEE transactions on automatic control*, **40:10**, pp. 1793–1796.

Engelsone, A., S. L. Campbell, and J. T. Betts (2007): “Direct transcription solution of higher-index optimal control problems and the virtual index.” *Applied numerical mathematics*, **57**, pp. 281–296.

Fornbern, A. (1998): *A Practical Guide to Pseudospectral Methods*. Cambridge University Press.

Garg, D., W. W. Hager, and A. V. Rao (2011a): “Pseudospectral methods for solving infinite-horizon optimal control problems.” *Automatica*, **47**, pp. 829–837.

Garg, D., M. A. Patterson, C. Francolin, C. L. Darby, G. T. Huntington, W. W. Hager, and A. V. Rao (2011b): “Direct trajectory optimization and costate estimation of finite-horizon and infinite-horizon optimal control problems using a radau pseudospectral method.” *Computational optimization and application*, **49**, pp. 335–358.

Gong, Q., W. Kang, and I. M. Ross (2006): “A pseudospectral method for the optimal control of constrained feedback linearizable systems.” *IEEE transactions on automatic control*, **51:7**, pp. 1115–1129.

Gong, Q., I. M. Ross, and F. Fahroo (2009): “A Chebyshev pseudospectral method for nonlinear constrained optimal control problems.” In *Proc. of the 48th conference on decision and control*, pp. 5057–5062.

Gong, Q., I. M. Ross, and W. Kang (2007): "A unified pseudospectral framework for nonlinear controller and observer design." In *Proc. of the 2007 american control conference*, pp. 1943–1949.

Gong, Q., I. M. Ross, W. Kang, and F. Fahroo (2008): "Connections between the covector mapping theorem and convergence of pseudospectral methods for optimal control." *Computational optimization and application*, **41**, pp. 307–335.

Huntington, G. and A. Rao (2007): "A comparison between global and local orthogonal collocation methods for solving optimal control problems." In *American Control Conference, 2007. ACC '07*, pp. 1950 –1957.

Jarre, F. and J. Stoer (2004): *Optimierung*. Springer.

JModelica.org (2012): *version 1.8*. JModelica.org.

Kameswaran, S. and L. T. Biegler (2008): "Convergence rates for direct transcription of optimal control problems using collocation at radau points." *Computational optimization and application*, **41**, pp. 81–126.

Kang, W. (2008): "The rate of convergence for a pseudospectral optimal control method." In *Proc. of the 47th IEEE conference on decision and control*, pp. 521–527.

Kang, W. and N. Bedrossian (2007): "Pseudospectral optimal control theory makes debut flight, saves NASA 1M in under three hours." *SIAM News*, **40:7**.

Kang, W., Q. Gong, I. M. Ross, and F. Fahroo (2007): "On the convergence of nonlinear optimal control using pseudospectral methods for feedback linearizable systems." *International journal of robust and nonlinear control*, **17**, pp. 1251–1277.

Laird, C., A. Wong, and J. Åkesson (2011): "Parallel solution of large-scale dynamic optimization problems." In *21st European Symposium on Computer-Aided Process Engineering*.

Laird, C. D. and L. T. Biegler (2006): "Large-scale nonlinear programming for multi-scenario optimization."

Magni, L., D. Raimondo, and F. Allgöwer (2009): *Nonlinear Model Predictive Control: Towards New Challenging Applications*. Lecture Notes in Control and Information Sciences. Springer.

MATLAB (2010): *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts.

Polak, E. (1973): "An historical survey of computational methods in optimal control." *SIAM Review*, **15:2**, pp. 553–584.

Pontryagin, L. S., V. G. Boltyanskii, R. V. Gamkrelidze, and E. Mishchenko (1962): *The mathematical theory of optimal processes (International series of monographs in pure and applied mathematics)*. Interscience Publishers.

Rao, C. V., S. J. Wright, and J. B. Rawlings (1998): "Application of interior-point methods to model predictive control." *Journal of optimization theory and applications*, **99:3**, pp. 723–757.

Song, W. and S. J. Dyke (2011): "Application of pseudospectral method in stochastic optimal control of nonlinear structural systems." In *Proc. of the 2011 american control conference*, pp. 2504–2509.

Tanartkit, P. and L. T. Biegler (1996): "A nested, simultaneous approach for dynamic optimization problems - I." *Computers chemical engineering*, **6**, pp. 735–741.

Wächter, A. and L. T. Biegler (2006): "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming." *Mathematical Programming*, **106**, pp. 25–57.

Zhang, W. and M. Heping (2008): "The chebyshev-legendre collocation method for a class of optimal control problems." *International Journal of Computer Mathematics*, **85:2**, pp. 225–240.

Åström, K.-J. and B. Wittenmark (1996): *Computer-controlled systems*. Prentice Hall.