3. Finite-time optimal control

In this chapter, we will study finite-time optimal control problems. Such problems arise in many contexts, for example in planning of financial investments and in trajectory generation for motion control systems. Optimal control is also a natural framework for studying the limits of performance of dynamical systems. In the context of this course, finite-time optimal control is one of the fundamental building block for model-predictive control. The tools introduced in this chapter will also allow us to derive state feedback and estimator gains that are optimal for linear systems with respect to quadratic costs in the control and states.

We begin by considering optimal control problems over a finite *horizon* of N steps. In these problems, we are given a (possibly nonlinear) dynamical system

$$x_{t+1} = f(x_t, u_t) \tag{3.1}$$

with initial state x_0. Our aim is to find a control sequence $\{u_0, u_1, \dots u_{N-1}\}$ such that the cost

$$\sum_{t=0}^{N-1} g_t(x_t, u_t) + g_N(x_N)$$

is minimized along the corresponding trajectory $\{x_0, x_1, \dots, x_N\}$ of the dynamical system (3.1). In addition, the states and the controls may be subject to constraints, $x_t \in X_t$ and $u_t \in U_t$. For example, U_t may be the set of all controls which obey given magnitude constraints and X_t may be the set of states which satisfy prescribed operational limits. We will write these finite-time optimization problem on the following standard form

$$\begin{array}{lll} \underset{\{u_0,u_1,u_{N-1}\}}{\text{minimize}} & \sum_{t=0}^{N-1} g_t(x_t,u_t) + g_N(x_N) & \\ \text{subject to} & x_{t+1} = f_t(x_t,u_t) & t = 1,\dots,N-1 \\ & x_t \in X_t & t = 1,\dots,N-1 \\ & u_t \in U_t & t = 1,\dots,N-1 \end{array} \tag{3.2}$$

The formulation (3.2) looks for optimal open-loop sequences $\{u_0, u_1, \dots, u_{N-1}\}$. These may depend on the initial state but do not account for the actual evolution of the system state. It is typically much more difficult to compute optimal feedback policies, *i.e.* $u_t = \phi_t(x_t)$.

Optimal control problems on the general form (3.2) are hard to solve. We will therefore need to impose additional structure to ensure tractability. As we will see later in these notes, restricting the dynamics to be linear and the cost functions to be quadratic will allow us to derive very strong results. In general, however, we will not be able to find the optimal solution in closed form but have to solve these finite-time optimal control problems numerically. Appendix C contains a brief introduction to mathematical programming, which is useful to fully appreciate this chapter.

3.1 Least-squares estimation and energy-optimal state transfer

We begin with studying two important problems that admit analytical solutions. They are based on results for minimizing of convex quadratic functions with and without linear constraints.

Proposition 3.1.1 — Completion-of-squares lemma. All minimizers of the quadratic function

$$f(x) = x^T Px + 2q^T x + r$$

with $P \succeq 0$ satisfy the normal equations

$$Px + q = 0.$$

If $P \succ 0$, then the minimizer is unique and given by

$$x^\star = -P^{-1}q$$

with corresponding minimal value

$$f^\star = r - q^T P^{-1} q = r - (x^\star)^T P x^\star.$$

Moreover, $f(x)$ can be re-written as a completion-of-squares

$$f(x) = (x - x^\star)^T P (x - x^\star) + r - (x^\star)^T P x^\star.$$

The result follows immediately from the first-order optimality conditions for unconstrained convex optimization in Appendix C and the differentiation rules for quadratic forms in Appendix B. As an example of how we can use this result, let us consider a classical least-squares problem.

■ **Example 3.1 — Least-squares parameter estimation.** The motivation for the least-squares estimation problem comes from the problem of estimating a parameter from a set of inconsistent observations. We are then given a vector of observations $y \in \mathbb{R}^m$ and a model

$$y = C\theta$$

for some given matrix $C \in \mathbb{R}^{m\times n}$. Our tasks is to estimate the parameter vector $\theta \in \mathbb{R}^n$. Since the system of equations is inconsistent with the observations, there is no θ for which the model explains the observations perfectly. It is then natural to try to minimize the size of the residuals

$$r = y - C\theta.$$

If we measure the size of r by its squared Euclidean norm, then the optimal θ minimizes

$$\begin{aligned} f(\theta) &= r^T r = (y - C\theta)^T (y - C\theta) = \\ &= \theta^T C^T C\theta - 2y^T C\theta + y^T y \end{aligned}$$

Since $C^T C$ is positive semidefinite, $f(\theta)$ is convex and the optimal solution $\theta^\star$ has to satisfy the normal equations

$$C^T C\theta^\star - C^T y = 0$$

If $C^T C$ is invertible, then we can form the optimal solution explicitly as

$$\theta^\star = (C^T C)^{-1} C^T y$$

■

The least-squares problem typically appears when we have more observations than parameters to estimate. Let us now instead consider the opposite situation, *i.e.*, when we need to solve the system of m linear equations in $\theta \in \mathbb{R}^n$

$$C\theta = y$$

with $m < n$. Since there are then typically many θ that satisfy the constraints, it is natural to look for the smallest parameter vector that explains the observations. If we measure size of θ in terms of its Euclidean norm, we would like to find the θ that solves the optimization problem

$$\begin{array}{ll} \text{minimize} & \theta^T\theta \\ \text{subject to} & C\theta = y \end{array}$$

As the next result shows, this problem also admits a closed-form solution.

Proposition 3.1.2 Consider the linearly constrained quadratic program

$$\begin{array}{ll} \text{minimize} & x^T x \\ \text{subject to} & Cx = d \end{array}$$

where $C \in \mathbb{R}^{m\times n}$, $d \in \mathbb{R}^m$ and $m < n$. If $\text{rank}(C) = m$, then the optimal solution is

$$x^\star = C^T (CC^T)^{-1} d.$$

The result is proven in Appendix C. We will now use it to study energy-optimal state-transfer for linear systems. To this end, consider the reachable discrete-time linear system

$$x_{t+1} = Ax_t + Bu_t$$

with known initial state x_0. We are interested in finding an optimal input sequence which ensures that $x_N = x_{\text{tgt}}$ for some fixed planning horizon N and target state x_{tgt}. As we have already observed in Chapter 1, the prediction equations are linear in the control input:

$$x_N = A^N x_0 + \sum_{k=0}^{N-1} A^k B u_{N-1-k} := c_N + \mathcal{C}_N \begin{bmatrix} u_0 \\ \vdots \\ u_{T-1} \end{bmatrix} = c_N + \mathcal{C}_N U_N$$

where

$$\mathcal{C}_N = \begin{bmatrix} A^{N-1}B & A^{N-2}B & \dots & AB & B \end{bmatrix}, \qquad c_N = A^N x_0$$

We recognize $\mathcal{C}_N$ as the controllability matrix over horizon N and c_N as the free response of the system to the initial value $x(0) = x_0$ after N steps.

Our focus is on minimum-energy control, *i.e.* to find input sequences $\{u_0, \dots, u_{N-1}\}$ which minimize the total energy

$$E(N, x_{\text{tgt}}) = \sum_{k=0}^{N-1} u_k^2 = U_N^T U_N$$

For simplicity, we let $x_0 = 0$, so that $c_T = 0$. The minimum-energy state transfer problem can therefore be solved via the quadratic program

$$\begin{array}{ll} \text{minimize} & U_N^T U_N \\ \text{subject to} & \mathcal{C}_N U_N = x_{\text{tgt}} \end{array}$$

If the system is reachable and $N \geq n$, the controllability matrix will have full rank and by Proposition 3.1.2 the optimal solution is

$$U_N^\star = \mathcal{C}_N^T (\mathcal{C}_N \mathcal{C}_N^T)^{-1} x_{\text{tgt}}$$

We can also compute the optimal value of this problem (which now has the interpretation of the minimum energy cost):

$$E(N, x_{\text{tgt}}) = (U_N^\star)^T (U_N^\star) = x_{\text{tgt}}^T (\mathcal{C}_N \mathcal{C}_N^T)^{-1} x_{\text{tgt}} = x_{\text{tgt}} \left(\sum_{k=0}^{N-1} A^k B B^T (A^T)^k \right)^{-1} x_{\text{tgt}}$$

This expression has an interesting geometrical interpretation. The sets of states which are reachable in N steps with unit energy

$$\{x \mid x^T (\mathcal{C}_N \mathcal{C}_N^T)^{-1} x \leq 1\}$$

form ellipsoids centered at the origin. We make the discussion more concrete in the next example.

■ **Example 3.2** Consider the mechanical system shown in Figure 3.1 (left). A cart is connected via a spring and a damper to a wall and can be displaced by applying a force F. According to standard Newton mechanics, the displacement $p(t)$ can be described by the second-order ODE

$$m\ddot{p}(t) = F(t) - kp(t) - d\dot{p}(t).$$

Letting $m = 1$, $k = d = 0.1$ and $h = 1$, zero-order hold sampling gives the discrete-time model

$$x_{t+1} = \begin{bmatrix} 0.95 & 0.94 \\ -0.09 & 0.86 \end{bmatrix} x_t + \begin{bmatrix} 0.48 \\ 0.94 \end{bmatrix} u_t$$

In this discrete-time model, the first state is the displacement (position), the second state is the velocity, and the control signal is the applied force.

The set of states which are reachable with unit input for various horizon lengths N are shown in Figure 3.1 (right). The results correspond well with physical intuition: for small horizon lengths, we can displace the cart further if we do not need to drive it back to rest; for larger horizon lengths, the unit energy limitation makes it difficult to reach both far and have high speed at time N. ■

In practice, one may want to consider state-transfer problems with different objective functions and constraints. However, as constraints are added, it becomes increasingly difficult to find analytical solutions to the associated optimization problems. Instead, we have to resort to numerical computations. Next, we will show that when the system dynamics and constraints are linear and $g_t(x_t, u_t)$ is convex and quadratic in (x_t, u_t), we can solve finite-time optimal control problems efficiently and reliably using quadratic programming.

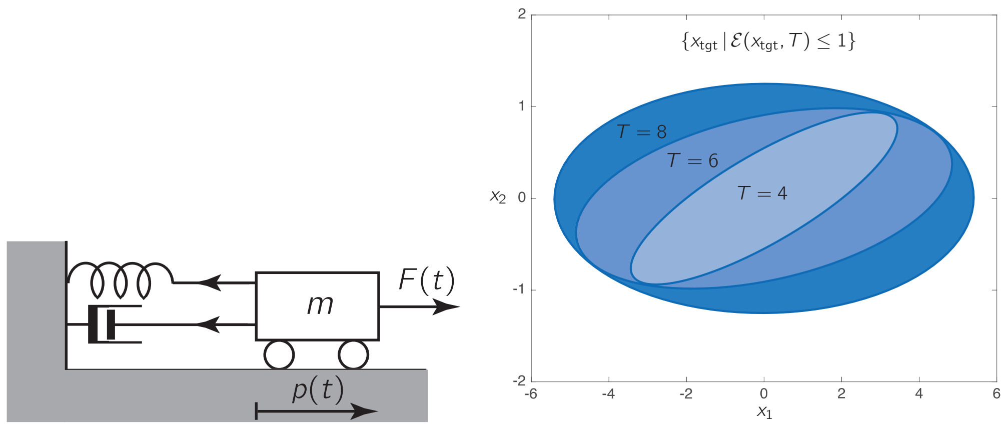


Figure 3.1: Mechanical system (left) and its unit-energy reachable sets for different time-horizons (right); here x_1 is the deplacement of the cart and x_2 its velocity.

3.2 Optimal open-loop control via quadratic programming

It is rare that we are able to solve finite-time optimal control problem explicitly, as we did for the minimum-energy state transfer problem. However, for many important control problems, we are able to compute the optimal control solution efficiently using numerical optimization. In this section, we will discuss optimal control problems on the form

$$\begin{array}{lll} \text{minimize} & \sum_{t=0}^{N-1} x_t^T Q_1 x_t + u_t^T Q_2 u_t + x_N^T Q_f x_N & \\ \text{subject to} & x_{t+1} = Ax_t + Bu_t & t = 0,\dots,N-1 \\ & M^x x_t + M^u u_t \le m & t = 0,\dots,N-1 \\ & M_N x_N \le m_N & \end{array} \tag{3.3}$$

Thus, we restrict ourselves to discrete-time linear systems with quadratic cost functions and linear constraints on the states and controls. The constraints

$$M^x x_t + M^u u_t \le m$$

model the state and control constraints at each sampling instant. If we have no control constraints, then $M^u = 0$, and if we have no state constraints, then $M^x = 0$. The most common control constraint is the magnitude limitation on the controls,

$$|u_t| \le u_{\max}$$

which we can express as the two linear inequalities $u_t \le u_{\max}$ and $-u_t \le u_{\max}$, *i.e.*

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} u_t \le \begin{bmatrix} u_{\max} \\ u_{\max} \end{bmatrix}$$

Similarly, the constraint that an output signal

$$y_t = Cx_t + Du_t$$

should lie between upper and lower bounds $y_{\min} \le y_t \le y_{\max}$, can be expressed as

$$\begin{bmatrix} C \\ -C \end{bmatrix} x_t + \begin{bmatrix} D \\ -D \end{bmatrix} u_t \le \begin{bmatrix} y_{\max} \\ -y_{\min} \end{bmatrix}.$$

The combination of control magnitude and output constraints also fits the standard form with

$$\begin{bmatrix} 0 \\ 0 \\ C \\ -C \end{bmatrix} x_t + \begin{bmatrix} 1 \\ -1 \\ D \\ -D \end{bmatrix} u_t \le \begin{bmatrix} u_{\max} \\ u_{\max} \\ y_{\max} \\ -y_{\max} \end{bmatrix}.$$

We will explore some additional constraints in the exercises. Finally, the terminal state constraint

$$M_N x_N \le m_N$$

is used to express the requirement that the terminal state lies in a polyhedron $\mathscr{P} = \{x \,|\, M_N x \le m_N\}$. If we try to achieve a fixed target state, $\mathscr{P} = \{x_{\text{tgt}}\}$, we can do so by letting

$$M_N = \begin{bmatrix} I \\ -I \end{bmatrix}, \qquad m_N = \begin{bmatrix} x_{\text{tgt}} \\ -x_{\text{tgt}} \end{bmatrix}. \tag{3.4}$$

3.2.1 Quadratic programming

A *quadratic program (QP)* is an optimization problem that involves minimizing a convex quadratic function subject to linear constraints:

$$\begin{array}{lll} \underset{z\in\mathbb{R}^n}{\text{minimize}} & z^T P z + 2q^T z + r & \\ \text{subject to} & a_i^T z \le b_i, & i = 1,\dots,m \\ & g_i^T z = h_i, & i = 1,\dots,p \end{array} \tag{3.5}$$

In order for a QP to be convex, we have to require that $P \succeq 0$. If $P = 0$, the problem reduces to a *linear program*, which we will discuss briefly later in these notes. Both linear and quadratic programming problems have been studied extensively and admit a rich and useful theory. In addition, the numerical solvers for these problems have reached a high level of maturity and can routinely solve problems with millions of variables and constraints.

Note that our standard form only has one-sided inequalities, but if our problem also has inequalities on the form $a^T z \ge b$ we simply use the observation

$$a^T z \ge b \Leftrightarrow -a^T z \le -b.$$

to transform them into the standard form. In principle, we could also have removed the equality constraints from (C.7), since $g^T z = h \Leftrightarrow (g^T z \le h) \wedge (g^T z \ge h) \Leftrightarrow (g^T z \le h) \wedge (-g^T z \le -h)$. However, we decided to keep the equality constraints since most modern solvers perform better when they are informed about them.

If we compare the QP formalism (3.5) with our finite-time optimal control problem (3.3), we notice that we can pose the optimal control problem as a QP with decision variables

$$z = \begin{pmatrix} u_0, & \dots & , u_{N-1}, & x_1, & \dots, & x_N \end{pmatrix}.$$

We will perform this transformation in detail below. However, there is a wealth of domain-specific languages for optimization that perform this transformation automatically. The next example shows how we can use the modeling language YALMIP [12] to solve the energy-optimal state transfer problem under control constraints as a QP.

■ **Example 3.3** Let us return to the mechanical system studied in Example 3.2. Our analysis showed that it is possible to transfer the state from x_0 to $x_{\text{tgt}} = [3, 1.275]$ with unit energy in four samples (the state is on the boundary of the dark blue ellipsoid in Figure 3.1). However, that analysis does not account for magnitude limitations on the control. In the quadratic programming framework, it is easy to add such constraints. The listing in Figure 3.3 shows YALMIP code for solving the minimum energy problem with the control constraints $|u_t| \leq u_{\max}$.

If we let $u_{\max} = 0.5$ and $N = 4$, the associated optimization problem is infeasible, *i.e.* it is no longer possible to execute the state transfer in 4 time steps with the magnitude limitations on the control. We need to extend the horizon to $N = 6$ before the associated QP has a solution. ■

```
% System matrices and problem specification
A=[0.95, 0.94; -0.09 0.86]; B=[0.48;0.94]; n=2; m=1;
 x_init=[0;6]; N=10;

% Declaration of decision variables
for t=1:N,
     x{t}=sdpvar(n,1); u{t}=sdpvar(m,1);
end;

% Definition of objective and constraints
objective=0;
constraints=[x{1}==x_init];
for t=1:N-1,
     objective=objective+u{t}'*u{t};
     constraints=[constraints, x{t+1}==A*x{t}+B*u{t}, -1<= u{t} <=1];
end;
constraints=[constraints, x{N}==[0;0]];

% Solve optimization problem
optimize(constraints,objective);
```

Figure 3.2: YALMIP code for finite-time energy optimal state transfer with magnitude constraints on the control.

3.2.2 The extensive and condensed formulations for finite-time optimal control

Although we can often rely on algebraic modeling languages to transform a problem into the standard form for QPs accepted by modern solvers, it is instructive to understand the principles of how this transformation is done. Not only does this exercise confirm that it is possible to formulate the optimal control problem as a QP, but it also gives insight into the size and structure of the corresponding matrices, as well as the different modeling options.

Since we restrict ourselves to discrete-time linear systems, the predicted state at time t is a linear function of the initial states and previous controls,

$$x_t = \begin{bmatrix} B & AB & \cdots & A^{t-1}B \end{bmatrix} U_t + A^t x_0$$

Thus, if we introduce $X_N \in \mathbb{R}^{n\times N}$ and $U_N \in \mathbb{R}^{m\times N}$ as

$$X_N = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \qquad U_N = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}.$$

we can relate X_N and U_N by the linear equality constraints

$$X_N = \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & 0 & 0 \\ \vdots & \ddots & \ddots & 0 \\ A^{N-1}B & \dots & AB & B \end{bmatrix} U_N + \begin{bmatrix} Ax_0 \\ A^2x_0 \\ \vdots \\ A^N x_0 \end{bmatrix} := \mathcal{H}_N U_N + h_N.$$

Next, the state and control constraints can be collected into constraints on X_T and U_T:

$$\underbrace{\begin{bmatrix} M^x & 0 & \dots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & \ddots & M^x & 0 \\ 0 & \dots & 0 & M^N \end{bmatrix}}_{\mathcal{M}^x} X_N + \underbrace{\begin{bmatrix} M^u & 0 & \dots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & \ddots & M^u & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix}}_{\mathcal{M}^u} U_N \leq \underbrace{\begin{bmatrix} m \\ \vdots \\ m \\ m_N \end{bmatrix}}_{\mathcal{m}}$$

Finally, since the stage costs and final costs are quadratic

$$g(x_t, u_t) = x_t^T Q_1 x_t + u_t^T Q_2 u_t, \qquad q_N(x_N) = x_N^T Q_f x_N,$$

the total cost of the optimal control problem can be expressed as

$$X_N^T \begin{bmatrix} Q_1 & 0 & \dots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & \ddots & Q_1 & 0 \\ 0 & \dots & 0 & Q_N \end{bmatrix} X_N + U_N^T \begin{bmatrix} Q_2 & 0 & \dots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & \ddots & Q_2 & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix} U_N := x_N^T \mathcal{Q}^x X_N + U_N^T \mathcal{Q}^u U_N$$

which is convex if Q_1, Q_2 and Q_N all are positive semidefinite.

Collecting objectives and constraints, the finite-time optimal control problem can then be formulated as the following quadratic programming problem in X_N and U_N:

$$\begin{array}{ll} \text{minimize} & X_N^T \mathcal{Q}^x X_N + U_N^T \mathcal{Q}^u U_N \\ \text{subject to} & \mathcal{M}^x X_N + \mathcal{M}^u U_N \leq \mathcal{m} \\ & X_N - \mathcal{H}_N U_N = \hbar_N \end{array}$$

This formulation is typically referred to as the *extensive form*, since both X_N and U_N are decision variables. It is possible to express the same problem in a *condensed form* by using the equalities to eliminate X_N from the problem. In this case, the objective becomes

$$\begin{aligned} X_N^T \mathcal{Q}^x X_N + U_N^T \mathcal{Q}^u U_N &= (\mathcal{H}_N U_N + \hbar_N)^T \mathcal{Q}^x (\mathcal{H}_N U_N + \hbar_N) + U_N^T \mathcal{Q}^u U_N = \\ = & U_N^T (\mathcal{H}_N^T \mathcal{Q}_N \mathcal{H}_N + \mathcal{Q}_U) U_N + 2(\mathcal{H}_N^T \mathcal{Q}_x \hbar_N)^T U_N + \hbar_N^T \mathcal{Q}_x \hbar_N \end{aligned}$$

while the state constraints reduce to

$$\mathcal{M}^x (\mathcal{H}_N U_N + \hbar_N) + \mathcal{M}^u U_N = (\mathcal{M}^x \mathcal{H}_N + \mathcal{M}^u) U_N + \mathcal{M}^x \hbar_N \leq \mathcal{m}$$

The condensed-form QP thus takes the form

$$\begin{array}{ll} \text{minimize} & U_N^T (\mathcal{H}_N^T \mathcal{Q}_N \mathcal{H}_N + \mathcal{Q}_U) U_N + 2(\mathcal{H}_N^T \mathcal{Q}_x \hbar_N)^T U_N + \hbar_N^T \mathcal{Q}_x \hbar_N \\ \text{subject to} & (\mathcal{M}^x \mathcal{H}_N + \mathcal{M}^u) U_N \leq \mathcal{m} - \mathcal{M}^x \hbar_N \end{array}$$

While the extensive form has $(m+n) \times N$ variables, the matrices describing the problem are sparse. The condensed form, on the other hand, has only $m \times N$ variables but a dense description. The most appropriate formulation depends on the specific QP solver which will be used.

3.2.3 Example: time-optimal movement of liquid containers

In many industrial packaging processes, one needs to move a package filled with liquid from one position to another (for example, from the filling station to the sealing station). To maintain a high production rate, it is essential to carry out this movement as quickly as possible. However, the acceleration of a package induces a motion of the liquid known as slosh. If left uncontrolled, this sloshing could result in liquid spill, causing product loss and contamination of machinery and packages, as well as a possible inability to seal the package properly.

In the next few pages, will consider finite-time optimal control of a packaging machine, illustrated in Figure 3.3. The models and parameters are taken from the PhD thesis [9], which contains many more details about modeling and control of liquid slosh. Our focus will be on moving the open containers from the filling station to the sealing station in minimal time without creating any liquid spill. For a representative machine, the filling time and movement time are similar; so decreasing the movement time with, say, 10% would decrease the total production time by 5%, with a corresponding increase in production rate.

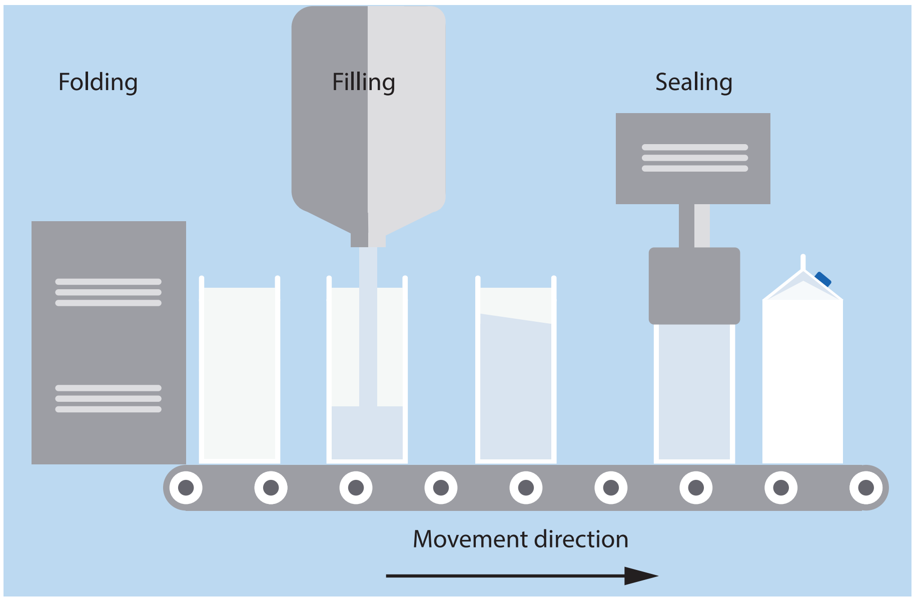


Figure 3.3: Schematic illustration of packaging machine. Packages are folded, filled with liquid and transported to a sealing machine. Our focus in this example is on the movement of the open liquid filled contained from the filling station to the sealing station.

The natural mathematical framework for modeling liquid movement is the Navier-Stokes partial differential equations. However, they are difficult to handle in our framework and we will instead use a simple linear ODE model to describe the liquid surface evaluation at the rear container wall, and the horizontal position of the cart. The model that we will use is

$$\dot{x}(t) = \begin{bmatrix} -2\zeta\omega_0 & -\omega_0 & 0 & 0 \\ \omega_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} \frac{a\omega_0}{2g} \\ 0 \\ 1 \\ 0 \end{bmatrix} u(t)$$

$$\begin{bmatrix} s(t) \\ p(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t) := \begin{bmatrix} C_s \\ C_p \end{bmatrix} x(t)$$

Here, we have defined the two outputs $s(t)$ for the surface evaluation at the rear of the package, and $p(t)$ for the container position on the conveyor belt.

We will consider a liquid container of height $h = 0.2$m, transported a distance of $L = 0.2$m. The corresponding model parameters are $\zeta = 0$, $\omega_0 = 21$ and $a = 0.07$; the parameter $g = 9.81$ is the standard constant of gravity. We are interested in moving the liquid container from rest at $p = 0$

to rest at $p = L$ without causing any slosh. Specifically, we will require that

$$|s(t)| \leq s_{\max} \qquad \forall t$$

where $s_{\max} = 3.5$cm. In addition, the control input is limited

$$|u(t)| \leq u_{\max} \qquad \forall t.$$

The initial machine configuration has $u_{\max} = g$. We will use a sampling time of 5ms and zero-order-hold control inputs. This means that we can construct a corresponding discrete-time model using the formulas derived in Chapter 1. Although we are interested in minimizing the transfer time, we will start with the simpler problem with a fixed horizon N and try to minimize the total control energy. After zero-order hold sampling of the continuous-time dynamics, we can pose the corresponding finite-time optimal control problem

$$\begin{array}{lll} \text{minimize} & \sum_{t=0}^{N-1} u_t^2 & \\ \text{subject to} & x_{t+1} = Ax_t + Bu_t, & t = 0, \dots, N-1 \\ & |C_1 x_t| \leq s_{\max}, & t = 0, \dots, N-1 \\ & |u_t| \leq u_{\max}, & t = 0, \dots, N-1 \\ & x_N = x_{\text{tgt}} & \end{array}$$

Thus, it fits our standard form with $Q_1 = 0$, $Q_2 = I$ and $Q_f = 0$, while

$$M^x \begin{bmatrix} C_1 \\ -C_1 \\ 0 \\ 0 \end{bmatrix} =, \qquad M^u = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}, \qquad m = \begin{bmatrix} s_{\max} \\ -s_{\max} \\ u_{\max} \\ -u_{\max} \end{bmatrix}$$

and M_N and m_N given by (3.4). The problem is readily transformed into a QP on standard form.

The nominal transfer time for the package from its initial position to its target is $T = 0.5$ seconds, which leads to a planning problem over 100 sampling instances. The resulting quadratic problem is feasible and the optimal solution is shown in Figure 3.4.

We are interested in speeding up the movement, and would like to find the smallest horizon N for which we can solve the state transfer problem. We find N by a simple bisection search: we let $N_{\min} = 0$ and $N_{\max} = 0.5/h$. Then in each iteration, we try to solve the finite-time optimal control problem for $N_{\text{mid}} = \lfloor (N_{\min} + N_{\max})/2 \rfloor$. If the associated optimization problem is feasible, we let $N_{\max} = N_{\text{mid}}$, otherwise we set $N_{\min} = N_{\text{mid}}$. We then update N_{mid} and repeat, until we notice that $N_{\text{mid}} = N_{\min}$. The bisection search reveals that the minimum transfer time is $T = 0.38$s, a reduction of over 20% from the nominal solution. The cart position, liquid elevation, and the associated control input are shown in Figure 3.5.

We notice that in the time optimal state transfer, both the control and the slosh constraints are active during large periods of time. In fact, the optimal control has a intuitive interpretation: one first accelerates the cart until the positive slosh constraint becomes active and then adjust the input to keep the slosh at that level. This gives the maximum admissible acceleration of the cart. After an appropriate time, one then prepares the stopping by driving the system to the state where the negative slosh constraint is active and keeping it there (which results in the maximum admissible deceleration). Finally, all states are driven to rest at target position. In fact, this is exactly how time-optimal control solution for the continuous-time system works. A formal proof of this claim requires continuous-time optimal control theory, which is beyond the scope of this course. However, we can recover the time-optimal control solution by using a smaller sampling interval, see Figure 3.6 for state and control trajectories, and Figure 3.7 for a pictorial illustration of the container movement.

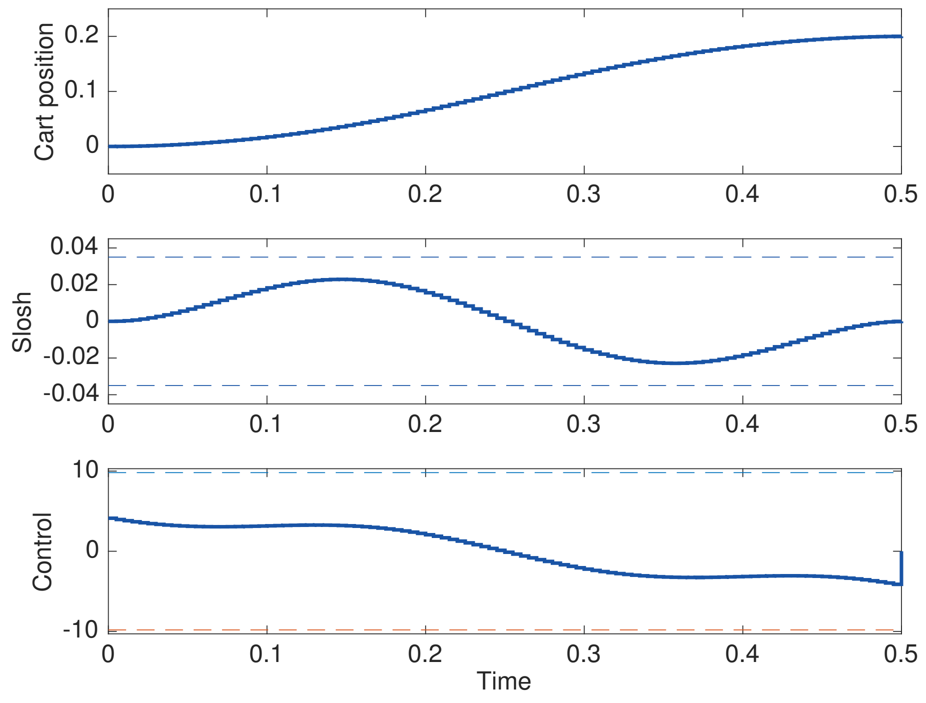


Figure 3.4: Minimum energy control of the cart with a nominal transfer time of 0.5 seconds. Neither control nor slosh constraints are active, indicating a potential for a more aggressive control.

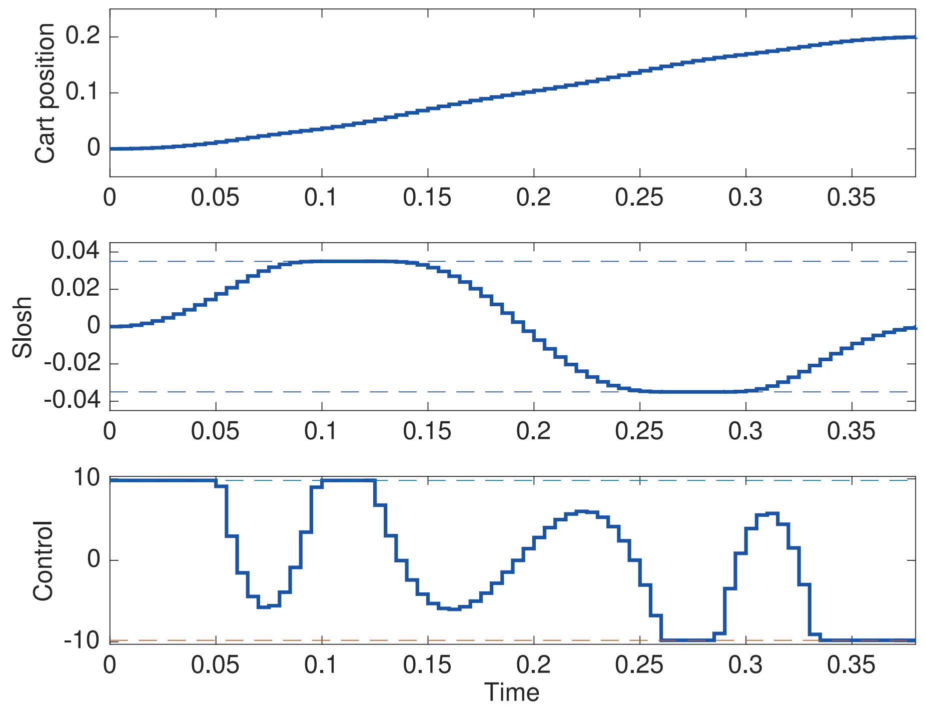


Figure 3.5: In the minimum time solution, both control and slosh constraints are active during large periods of time.

It is natural to ask if we could perform the transfer faster with a stronger motor (larger $u_{\max}$), or if we could use a weaker (and perhaps cheaper) motor without significant performance degradations. It turns out that no dramatic performance improvements are obtained for $u_{\max}$ above g, see Figure 3.8.

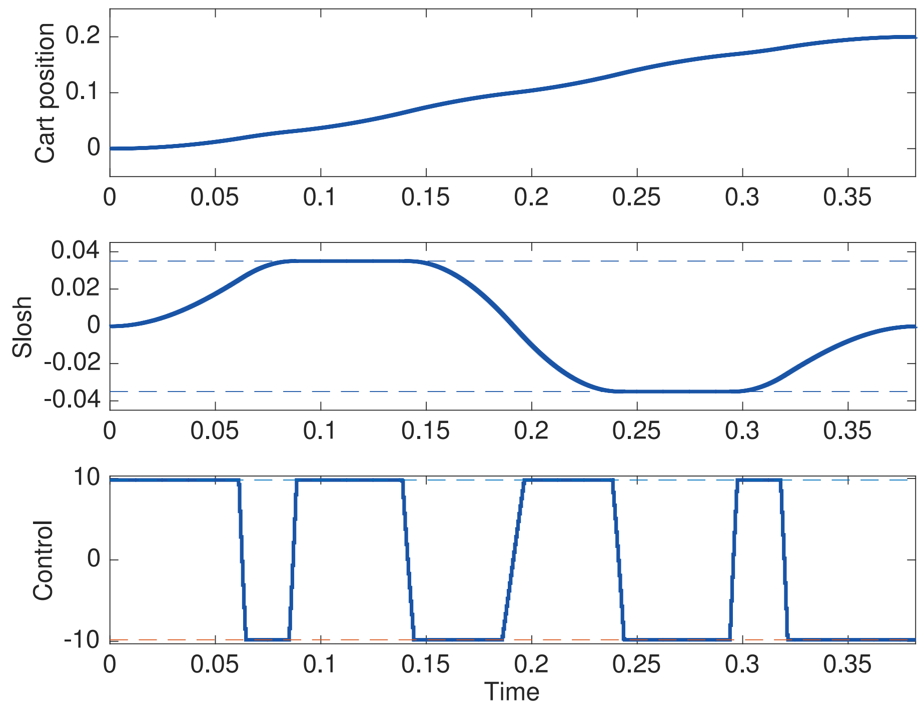


Figure 3.6: With a shorter sampling interval, the optimal input approaches the “bang-bang” nature of time-optimal control of continuous-time systems.

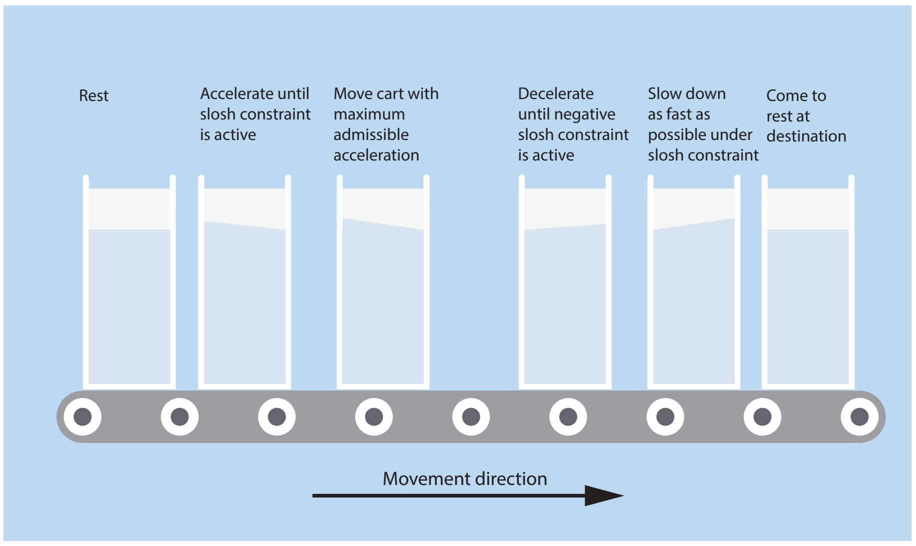

Rest
Accelerate until slosh constraint is active
Move cart with maximum admissible acceleration
Decelerate until negative slosh constraint is active
Slow down as fast as possible under slosh constraint
Come to rest at destination
Movement direction

Figure 3.7: Time-optimal movement strategy in pictures.

The reason for this is that the slosh constraint limits the transfer time. The control required to have $s(t) = s_{\max}$ and $\dot{s}(t) = 0$ is exactly $u(t) = g$.

3.3 Optimal feedback control via dynamic programming

Dynamic programming (DP) is a powerful paradigm for transforming a complex optimization problem into a sequence of simpler subproblems. It can be understood intuitively by the following example. Imagine that you draw the shortest path between two points A and B on a sheet of paper.

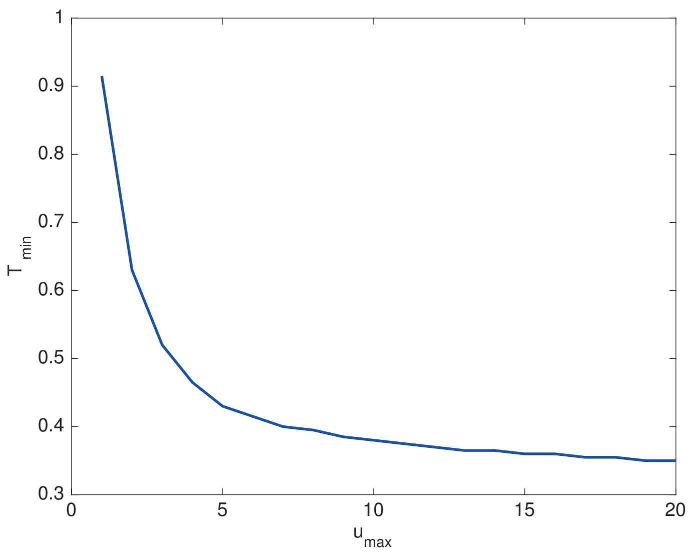


Figure 3.8: Minimum transfer time as function of the optimal control input $u_{\max}$. No significant reductions in the transfer time are possible when $u_{\max}$ exceeds g.

If you stop your pen at any intermediate point C along this path, the remaining path from C to B must be a shortest path. If this is not immediately clear to you, think about what would happen otherwise: it would then be better to deviate from the original path at C, which would contradict its optimality. This intuitive observation is a variation of the more general *principle of optimality*

> Any optimal policy has the property that, whatever the current state and decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the current decision.

The principle of optimality will allow us to compute optimal control laws in a recursive manner, often leading to closed-loop policies rather than just open-loop control sequences. Before we formalize the dynamic programming approach further, let us continue to develop our intuition by considering a few examples.

Example: computing the shortest path in a graph.

Figure 3.9 shows the travel times by car for popular routes between some of Sweden's largest cities. The nodes in the graph represent cities, and arcs correspond to route choices. The numbers on each arc quantify the estimated travel time in minutes.

We are interested in finding the fastest route that takes us from M (Malmö) to S (Stockholm). To this end, we will use the observation that we made earlier: from any intermediate city on the optimal route, we must follow a fastest path to S. Thus, there is no need to keep track of all possible paths, only the shortest paths from every city to S. In fact, we will not even need to keep track of the shortest paths, only the shortest travel time (the "cost to go") from every city to S. With this information, we can determine the shortest path from M to S recursively: we start at M and drive to a neighboring city for which the travel time to that city plus the remaining "cost to go" is minimal. From this node, we then repeat the procedure: drive to the neighboring node with the smallest value of travel time plus "cost to go". We continue like this until we have reached our destination node S.

To determine the cost-to-go, we will proceed iteratively and compute the vector v_n, whose entries represent the shortest travel time that traverses n or less arcs. We use $+\infty$ to mark that S is not reachable in n or less arc traversals. Clearly, the only finite entry in v_0 (no arc traversals) is the one corresponding to S itself. Moving on to v_1, we see that we can reach S from 3 cities (V, Ö

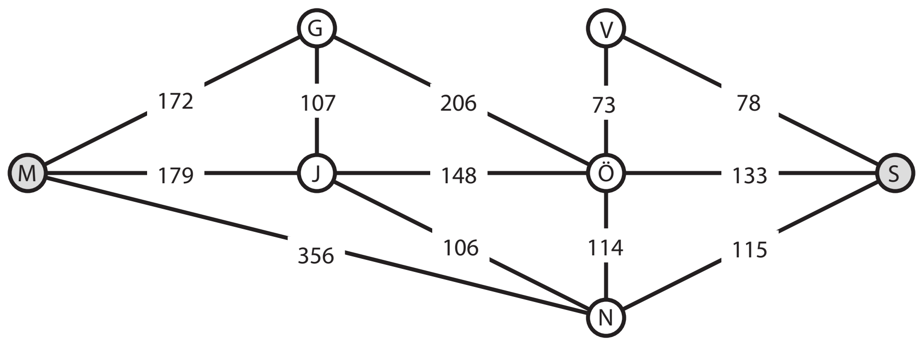


Figure 3.9: Travel times (minutes) by car between some of Sweden's largest cities.

and N) by traversing a single arc. We thus record the corresponding direct distances in v_1. Things become a little more interesting when we allow for up to two arc traversals. For example, it is now possible to reach S from Ö in three different ways (the direct route, via V, or via N). Letting $\tau[A,B]$ denote the travel time on the arc connecting cities A and B, we can now compare the three options.

$$v_2[\ddot{O}] = \min\left\{v_1[\ddot{O}], \tau[\ddot{O},V] + v_1[V], \tau[\ddot{O},N] + v_1[N]\right\}.$$

We note that the direct route is the fastest, hence $v_2[\ddot{O}] = v_1[\ddot{O}]$. We also note that it is now possible to find a route from M to S (namely via N). The corresponding shortest distance is

$$v_2[M] = \tau[M,N] + v_1[N] = 471$$

Now, moving on to v_3, we have more options for reaching S from M, since all of its neighbors can reach S in two arc traversals or less. Specifically,

$$\begin{aligned} v_3[M] &= \min\{v_2[M], \tau[M,G] + v_2[G], \tau[M,J] + v_2[J], \tau[M,N] + v_2[N]\} = \\ &= \tau[M,J] + v_2[J] = 400. \end{aligned}$$

We have thus found a new shorter route from M to S. After completing all the entries of v_3, we move on to v_4 only to notice that it is identical to v_3. Thus, we have now found the shortest travel times from all cities to S.

Table 3.1: Caption for the table.

city	v_0	v_1	v_2	v_3
S	0	0	0	0
V	$+\infty$	78	78	78
Ö	$+\infty$	133	133	133
N	$+\infty$	115	115	115
G	$+\infty$	$+\infty$	339	339
J	$+\infty$	$+\infty$	221	221
M	$+\infty$	$+\infty$	471	400

Although v_3 only contains the shortest distances, we can use it together with the arc costs to determine the optimal path. Since

$$v_3[M] = v_2[J] + \tau[M,J]$$

we should first drive to J. Then, since

$$v_2[J] = v_1[N] + \tau[J,N]$$

we should continue from J to N. Finally, as

$$v_1[N] = v_0[S] + \tau[N,S]$$

we should take the direct route from N to S.

This simple example displays some typical characteristics of dynamic programming problem. A complex decision problem (the fastest path to S) can be structured as a sequence of smaller subproblems (the fastest path to S traversing at most n arcs). The value of being in city C at stage n can be characterized by its optimal cost-to-go, v_n. This function can be computed recursively and the optimal action is guided by current cost and cost-to-go from next state.

Let us continue by solving a problem of a slightly different flavor.

Example: using the minimum number of coins to settle a debt.
Given coins valued at 1, 2 and 5 SEK, how can we determine the minimal number of coins necessary to pay a given amount?

We can notice that the dynamic programming principle is at play: if you have found the coins necessary to settle your debt in an optimal way and pay one of these, the remaining coins must be an optimal way to settle the rest of what you owe. It therefore makes sense to keep track of the vector v_n whose entries $v_n[i]$ are the minimal number of coins necessary to pay i SEK using at most n coins, see Table 3.2. As before, we use $+\infty$ to indicate that the corresponding amount cannot be paid using the allowed number of coins.

Table 3.2: Caption for the table.

amount	v_0	v_1	v_2	v_3
0	0	0	0	0
1	$+\infty$	1	1	1
2	$+\infty$	1	1	1
3	$+\infty$	$+\infty$	2	2
4	$+\infty$	$+\infty$	2	2
5	$+\infty$	1	1	1
6	$+\infty$	$+\infty$	2	2
7	$+\infty$	$+\infty$	2	2
8	$+\infty$	$+\infty$	$+\infty$	3
9	$+\infty$	$+\infty$	$+\infty$	3

For notational simplicity, we begin with v_0 (the amounts that we can settle without any coins). Next, v_1 has entries $+1$ at the positions corresponding to the denomination of the coins. To fill in v_2, we note

$$\begin{aligned} v_2[i] &= \min\{v_1[i], v_1[i-1]+1, v_1[i-2]+1, v_1[i-5]+1\} = \\ &= \min\{v_1[i], 1+\min\{v_1[i-1], v_1[i-2], v_1[i-5]\}\} \end{aligned}$$

In words, this means that to settle i SEK with at most 2 coins, we should settle it with 1 coin if possible, or use the best of the three possibilities that the different coin denominations offer. The computation of v_n for $n \geq 3$ is analogous.

One may ask if we really need to solve this problem using dynamic programming? Couldn't we simply use a greedy algorithm which chooses the coin of largest value smaller or equal to what remains to be paid, and repeats until we have paid the full amount? It turns out that the greedy solution is indeed optimal for this specific problem instance but not for general coin values. For example, if we were given coins valued at $\{1,4,5\}$, then the greedy approach would suggest to settle 8 SEK using four coins (one 5 SEK coin and three 1 SEK coins) while we could settle it more efficiently with two 4 SEK coins.

3.3.1 The dynamic programming formalism

The essential feature of dynamic programming is the structuring of the optimization problem into multiple *stages*, which are solved sequentially one stage at a time. Each one stage problem helps to define the characteristics of the next. In systems and control problems, the stage index typically corresponds to a time period in a planning horizon. In other problems, the stage index simply corresponds to problem size (as increasingly large decision problems are solved based on the solution of smaller ones). Associated with each stage is a system *state*. The state captures all information required to assess the consequences that the current *decision* has on future actions. The solution to an N-stage decision problem is then constructed by first solving a one-stage problem and then sequentially adding one stage at a time, computing the cost-to-go over increasingly longer horizons until the overall solution has been found. The evolution of stages relative to physical time (when present) can result in a *forward* or *backward induction process*.

To formalize the approach, we will consider dynamical systems on the form

$$x_{n+1} = f_n(x_n, u_n)$$

Here n is the stage index, x_n is the system state and u_n is the (control) action. Note that the transition function $f_n(x_n, u_n)$ can be stage-dependent. Our aim is to find a control policy, *i.e.* a rule $u_n = \mu_n(x_n)$ for how to choose u_n based on x_n, which minimizes the additive cost

$$J(u_0, \dots, u_{N-1}; x_0) = g_N(x_N) + \sum_{n=0}^{N-1} g_n(x_n, u_n)$$

We will refer to $g_n(x_n, u_n)$ as *stage costs* and $g_N(x_N)$ as *final cost*. The action may be further subject to *control constraints*, which we write as $u_n \in \mathcal{U}_n(x_n)$.

Let $\mu^\star = \{\mu_0^\star, \mu_1^\star, \dots, \mu_{N-1}^\star\}$ be an optimal policy for the basic problem. Consider the subproblem where we are in state x_k at stage k and would like to minimize the "cost-to-go"

$$g_N(x_N) + \sum_{n=k}^{N-1} g_k(x_k, \mu(x_k))$$

By the *principle of optimality*, the truncated policy $\{\mu_k^\star, \dots, \mu_N^\star\}$ is optimal for this subproblem. This leads to the dynamic programming algorithm and its optimality:

Theorem 3.3.1 — The DP algorithm. For every initial state x_0, the optimal cost $J^\star(x_0)$ of the basic problem is equal to $v_0(x_0)$, given by the last step of the following algorithm, which proceeds backwards from stage $N-1$ to stage 0:

$$v_N(x_N) = g_N(x_N) \tag{3.6}$$

$$v_k(x_k) = \min_{u_k \in \mathcal{U}_k(x_k)} \{g_k(x_k, u_k) + v_{k+1}(f_k(x_k, u_k))\} \qquad k = N-1, \dots, 0 \tag{3.7}$$

If $u_k^\star = \mu_k^\star(x_k)$ minimizes the right-hand-side of (3.7) for each x_k and each k, then the policy $\mu^\star = \{\mu_0^\star, \dots, \mu_{N-1}^\star\}$ is optimal.

The function $v_k(x_k)$ describes the optimal cost for an $N-k$ stage problem starting at state x_k in stage k and proceeding until stage N. The quantity $v_k(x_k)$ is called the *cost-to-go* at state x_k and stage k, and the function v_k as the *cost-to-go* function in stage k.

■ **Example 3.4 — fastest path problem.** Here, stages correspond to the maximum number of node traversals. States are nodes in the graph and the control is the decision which edge to traverse next. We notice that we have a state-dependent control constraints: we can only choose from edges that are outgoing from the current node. The transition function maps the current node to the node at the end of the selected edge, and $g_n(x_n,u_n)$ is the time required to traverse the edge selected by u_n, and 0 if we choose to stay in the current node. ■

■ **Example 3.5 — minimum coin settlement problem.** The stage index corresponds to the maximal number of coins used, and the state is the sum settled so far. The action is the choice of coin denomination to use, and the transition function adjusts the settled amount accordingly. The stage cost is 1 if we use a coin, and zero otherwise. ■

■ **Example 3.6** Consider the following sequential decision-making problem

$$\begin{array}{ll} \underset{u_1,\dots,u_T}{\text{maximize}} & \sum_{t=1}^{T} r_t(x_t,u_t) \\ \text{subject to} & x_{t+1} = x_t - u_t \end{array}$$

where $r_t = d^t(pu_t - u_t^2/x_t)$. The problem is motivated by fair pricing of a T-year lease of an underground mine: the state x_t represents the amount of ore left in the mine at the beginning of year t, and u_t the amount of ore extracted during that year. Selling the ore gives a profit of px_t but the extraction cost is u_t^2/x_t to model that extraction becomes more costly as the mine is depleted. The discount factor $d = 1/(1+r)$ where r is the current interest rate. In this way, the optimal value of the problem equals the net present value of the lease, *i.e.* the amount of money you would need to put in the bank today to be able to replicate the income stream that can be generated from the lease.

To solve the problem using dynamic programming, we consider the final year $t = T$, in which the optimal action is to maximize $r(x_T,u_T)$. The first-order optimality conditions yield

$$\mu_T^\star(x_T) = \frac{p}{2}x_T := L_T x_T$$

for which the net present value of the optimal operating cost is

$$v_T(x) = d^T r(x_T,\mu_T^\star(x)) = d^T L_T^2 x := d^T \alpha_T x$$

Next, we consider year $T-1$, for which we need to maximize

$$r_{T-1}(x_{T-1},u_{T-1}) + v(x_T) = r_{T-1}(x_{T-1},u_{T-1}) + v(x_{T-1}-u_{T-1}) =$$
$$= d^{T-1}\left(pu_{T-1} - \frac{u_{T-1}^2}{x_{T-1}}\right) + d^T\alpha_T(x_{T-1}-u_{T-1})$$

The first-order optimality conditions yield

$$u_{T-1}^\star(x_{T-1}) = \frac{1}{2}(p - d\alpha_T)x_{T-1} := L_{T-1}x_{T-1}$$

Inserting this control in the expression for the cost-to-go gives

$$v_{T-1}(x) = d^{T-1}\left(\frac{(p-d\alpha_T)^2}{4} + d\alpha_T\right)x := \alpha_{T-1}x$$

We can continue backward, computing v_{T-2}, v_{T-3}, etc. Recognizing that the same algebra applies in each step, we find that the optimal action at stage k is $u_k = L_k x_k$ and the cost-to-go has the form $v_k(x) = \alpha_k x$ where

$$\begin{aligned}\alpha_k &= d^k\left(\frac{(p-d\alpha_{k+1})^2}{4} + d\alpha_{k+1}\right)\\ L_k &= \frac{1}{2}(p-d\alpha_k)\end{aligned}$$

■

Dynamic programming applies both to systems with discrete state (such as the shortest path or debt settlement problems) or continuous state (as in the mining lease problem). Whether or not DP will be a useful technique depends on our ability to solve the one-stage problems efficiently, and the possibility to effectively represent the optimal value function. Ideally, the data structure used to represent the value function should not grow in complexity as the number of stages is increased.

3.3.2 Solving dynamic programming problems numerically

A drawback with the dynamic programming approach described so far is that we need to find a representation of the value function that can be propagated from one stage to the next. In many cases, it is difficult or impossible to find a convenient analytical expression of the value function. It is then natural to resort to numerical computations.

The simplest numerical approach to solving dynamic programming problems over a finite horizon is to define a grid on the state space and tabulate the value function at these grid points. A conceptual algorithm is shown below.

Algorithm 1 Numerical solution to finite-time DP

```
1: Define grid $\mathcal{X}_G = \{x^{(1)}, x^{(2)}, \dots, x^{(G)}\}$ of test point to cover the state space
2: Initialize $v_N[i] = g_N(x^{(i)})$ for all $x^{(i)} \in \mathcal{X}_G$.
3: for $k = N-1, N-2, \dots, 1$ do
4:     for $i = 1, 2, \dots, G$ do
5:         $v_k[i] = \min_{u \in \mathcal{U}_k(x^{(i)})} \{g_k(x^{(i)}, u) + \hat{v}_{k+1}(f_k(x^{(i)}, u))\}$
```

Since the next state $x_{k+1} = f_k(x_k, u_k)$ is not guaranteed to coincide with any of the grid points, we define $\hat{v}_{k+1}(x)$ as the approximation of $\hat{v}_{k+1}$, given its tabulated values $v_{k+1}[i]$. The simplest way to interpolate may be using linear or multi-linear interpolation, but many options exist. One also needs to pay attention to edge effects that occur when there is no feasible control that can keep the next state inside the gridded part of the state space.

The minimization on line 5 can either be done numerically, or by gridding of the possible values of u and finding the one which results in the lowest value of $v_k[i]$. Recall that $\mathcal{U}_k(x_k)$ accounts for both the fact that $u_k \in U_k$ and that $x_{k+1} = f(x_k, u_k) \in X_{k+1}$. If $\mathcal{U}_k(x^{(i)})$ is empty, we let $v_k[i] = +\infty$. It can be convenient to store the optimal control for each grid point and stage in a separate table, and define a (hopefully only slightly suboptimal) control policy by interpolating this table.

■ **Example 3.7** Consider the mechanical system from Example 3.2 and assume that we want to minimize a quadratic cost over a time horizon of $N = 10$, defined by the stage costs

$$g_t(x_t, u_t) = x_t^T x_t + 10u_t^2 \text{ for } t = 0, 1, \dots, 9, \quad g_{10}(x_t) = x_t^T x_t.$$

Defining a unifom grid on $\|x\| \leq 5$ with 31 gridpoints in each direction, and solving the dynamic programming problem numerically using Algorithm 1 yields the optimal feedback and cost-to-go functions shown in Figure 3.10. The optimal control appears to be linear and the cost-to-go quadratic. In Chapter 4, we will *prove* that this observation is indeed true, and that the optimal control law can be computed much more efficiently than by gridding of the state-space. If we add the constraint that the state must lie within the grid and the controls must have magnitude less than 5, we find the optimal control in Figure 3.10 (bottom left). Clearly, the optimal policy is no longer linear. Similarly, the cost-to-go function is no longer quadratic (same figure, bottom right).

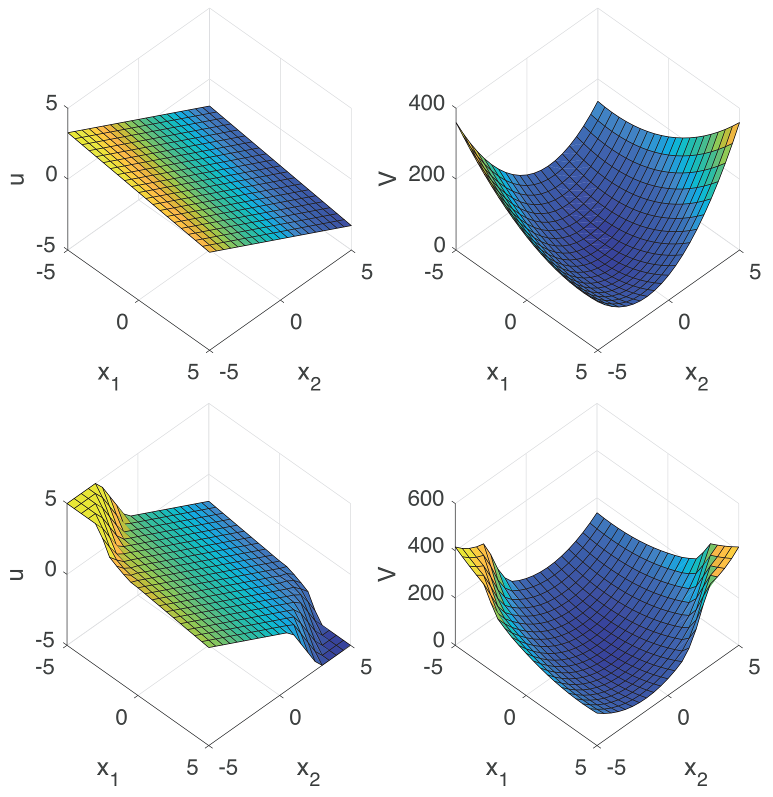


Figure 3.10: Optimal feedback policies (left column) and associated cost-to-go functions (right column). The upper row is for a linear system with quadratic cost: the optimal feedback is linear and the cost-to-go is quadratic. The lower row is for a linear system with state and control constraints: the optimal policy is now non-linear and the cost-to-go no longer quadratic.

■

3.3.3 Dynamic programming as optimization of multi-stage functions

While the dynamic programming formalism developed so far is elegant, it is not uncommon to find problems that are difficult or inconvenient to transform into the standard form. In these cases, it is useful to have an alternative understanding of dynamic programming as a means for minimizing multistage problems

$$J_N(x_1,\dots,x_N;x_0) = g_0(x_0,x_1) + g_1(x_1,x_2) + \cdots + g_{N-1}(x_{N-1},x_N) + g_N(x_N).$$

To build up some intuition, let us consider

$$J_2(x_1,x_2;x_0) = g_0(x_0,x_1) + g_1(x_1,x_2) + g_2(x_2)$$

Due to the structure of the problem, we can re-write the optimal value

$$J_2^\star(x_0) = \min_{x_1,x_2} J_2(x_1,x_2;x_0) = \\ = \min_{x_1}\left\{ g_0(x_0,x_1) + \min_{x_2}\{g_1(x_1,x_2)+g_2(x_2)\}\right\}$$

In the inner minimization problem,

$$\underset{x_2}{\text{minimize}} \quad g_1(x_1,x_2)+g_2(x_2)$$

we can view x_1 as a fixed parameter and derive an optimizer $x_2^\star(x_1)$ which depends on x_1. The optimal value of the inner minimization problem

$$v_1(x_1) = g_1(x_1,x_2^\star(x_1)) + g_2(x_2^\star(x_1))$$

is then a function of x_1, and the outer minimization problem takes the form

$$\underset{x_1}{\text{minimize}} \quad g_0(x_0,x_1)+v_1(x_1)$$

By solving the outer problem in a similar manner, we find $J_2^\star(x_0) = v_0(x_0)$.

Using an analogous argument, the optimal value of the N-stage problem $J_N^\star(x_0)$ equals $v_0(x_0)$ where the value functions $v_n(x_n)$ are computed recursively:

$$v_N(x_N) = g_N(x_N) \\ v_k(x_k) = \min_{x_{k+1}}\{g_k(x_k,x_{k+1}) + v_{k+1}(x_{k+1})\} \qquad n = N-1, N-2, \dots, 0$$

If we instead are given a multistage problem on the form

$$J_N(x_0,\dots,x_{N-1};x_N) = g_0(x_0)+g_1(x_0,x_1)+\dots+g_N(x_{N-1},x_N)$$

it is more natural to proceed using a forward induction

$$w_0(x_0) = g_0(x_0) \\ w_n(x_n) = \min_{x_{n-1}}\{g_n(x_{n-1},x_n)+w_{n-1}(x_{n-1})\}$$

The functions $w_n(x_n)$ have the interpretation of the smallest possible accumulated cost and is referred to as the *arrival cost* of stage n; $J_N^\star(x_N) = w_N(x_N)$.

3.3.4 A few words about the infinite-horizon case

In many control problems, there is no natural fixed terminal time, but we are rather interested in optimizing the performance over an infinite time-horizon. We will not cover such problems in detail in this course, as they are more technical to solve and would deviate our focus. Nevertheless, we will try to give high-level overview of some of the central ideas based on [5].

To this end, consider the infinite-horizon optimal control problem

$$\begin{array}{ll} \text{minimize} & \lim\limits_{N\to\infty} \sum_{t=0}^N g(x_t,u_t) \\ \text{subject to} & x_{t+1} = f(x_t,u_t) \\ & x_t \in X, \quad u_t \in U(x_t) \end{array}$$

with given initial state $x_0 \in X$. We are interested in finding feedback policies on the form $\pi = \{\mu_0,\mu_1,\dots\}$, where each μ_t is a function mapping X into a $U(x_t)$. We let Π denote the set of all such policies, and call π stationary if it has the form $\{\mu, \mu, \dots\}$. By applying a policy $\pi \in \Pi$ to the system, we generate state and control pairs $(x_t, \mu(x_t))$ for $t = 0, 1, \dots$ with cost

$$v_\pi = \lim_{N \to \infty} \sum_{t=0}^{N} g(x_t, \mu(x_t))$$

It turns out that if we do not impose additional conditions on the system dynamics or on the per-stage losses $g_t(\cdot,\cdot)$, the limit may not even exist [5]. In this short discussion, we will ensure existence by demanding that the per-stage losses are non-negative, *i.e.*, that

$$0 \le g(x_t, u_t) \le \infty. \tag{3.8}$$

(see, for example, [5] for other alternatives, such as considering average or discounted costs).

Now, we can view v_π as a function of $x \in X$ that takes values in $[0, \infty]$. We refer to it as the costfunction for π. The optimal cost function is

$$v^\star(x) = \inf_{\pi \in \Pi} v_\pi(x)$$

and a policy $\pi^\star \in \Pi$ is optimal if it attains the minimum of $v_\pi(x)$ for all $x \in X$, *i.e.*

$$v_{\pi^\star}(x) = \inf_{\pi \in \Pi} v_\pi(x) = v^\star(x), \qquad \forall x \in X.$$

Given our discussion for the finite-horizon case, one can expect that the optimal cost function $v^\star$ satisfies the *Bellman equation*

$$v^\star(x) = \inf_{u \in U(x)} \{g(x,u) + v^\star(f(x,u))\} \tag{3.9}$$

at that the stationary optimal policy can be found by minimizing the right-hand side of this equation. However, there is one subtlety, in the sense that if $\tilde{v}^\star$ satisfies the Bellman equation, then so does $\tilde{v}^\star + c$ for any positive constant c. It turns out that for our problem class, the optimal cost function is the smallest one which satisfies (3.9) (see [5] for detailed proofs.)

Theorem 3.3.2 Assume that the stage costs $g(x,u)$ satisfy (3.8). Then

(a) if $v^\star$ satisfies the Bellman equation (3.7) and $\tilde{v} : X \mapsto [0, \infty]$ also satisfies (3.7), then $v^\star \le \tilde{v}$.

(b) For all stationary policies π, we have

$$v_\pi(x) = g(x, \mu(x)) + v_\pi(f(x, \mu(x))) \qquad \forall x \in X$$

(c) A stationary policy $\mu^\star$ is optimal if and only if

$$\mu^\star(x) \in \arg\min_{u \in U(x)} \{g(x,u) + v^\star(f(x,u))\}$$

There are two principally different ways of finding the optimal stationary policy. The *value iteration* starts with some non-negative function $v_0 : X \mapsto [0, \infty]$ and generates a sequence $\{v_k\}$ via

$$v_{k+1}(x) = \inf_{u \in U(x)} \{g(x,u) + v_k(f(x,u))\}.$$

Policy iteration, on the other hand, starts with an admissible μ_0 and interleaves policy evaluations

$$v_{\mu_k}(x) = g(x, \mu_k(x)) + v_{\mu_k}(f(x, \mu_k(x)))$$

with policy improvements

$$\mu_{k+1}(x) \in \arg\min_{u \in U(x)} \left\{g(x,u) + v_{\mu_k}(f(x,u))\right\}$$

Both the policy and the value iterations are well-behaved and converge under some additional (and for many real-world systems mild) conditions, see [5] for details.

B. Quadratic forms

In this appendix, we will review some properties of quadratic forms $f : \mathbb{R}^n \mapsto \mathbb{R}$ where

$$f(x) = \sum_{i=1}^{n} \sum_{j \geq i}^{n} a_{ij} x_i x_j = \sum_{i=1}^{n} a_{ii} x_i^2 + \sum_{i=1}^{n} \sum_{j>n}^{n} a_{ij} x_i x_j.$$

It is convenient to represent such functions in matrix form

$$f(x) = x^T Q x$$

where

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad Q = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{12} & q_{22} & & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{1n} & q_{2n} & \cdots & q_{nn} \end{bmatrix}.$$

Note that

$$x^T Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j = \sum_{i} q_{ii} x_i^2 + \sum_{i=1}^{n} \sum_{j>i}^{n} (q_{ij} + q_{ji}) x_i x_j$$

Hence, the two parameterizations of the quadratic form match if we let $q_{ii} = a_{ii}$ and $q_{ij} = q_{ji} = a_{ij}/2$. This means that we can always assume that Q is a symmetric matrix, $Q = Q^T$ (where Q^T denotes the transpose of the matrix Q).

■ **Example B.1** The quadratic form

$$f(x) = x_1^2 - 2x_1 x_2 + 4x_2^2$$

can be written as

$$f(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 1 & -1 \\ -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

■

If we are given $f(x) = x^T Mx$ where $M \in \mathbb{R}^{n\times n}$ is not symmetric, then we can write it as $f(x) = x^T Qx$ where $Q = \frac{1}{2}(M + M^T)$ since

$$(q_{ij} + q_{ji}) = (m_{ij} + m_{ji}) \quad \text{for all } i, j$$

which implies that $x^T Qx = x^T Mx$.

■ **Example B.2** The quadratic form

$$f(x) = x_1^2 - 2x_1x_2 + 4x_2^2$$

can be written as

$$f(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 1 & -2 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} := x^T Ax$$

It can also be written as $x^T Qx$ with $Q = (A + A^T)/2$, i.e. as

$$f(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 1 & -1 \\ -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

■

B.1 Gradients and Hessians of quadratic forms

When performing optimization of functions of several variables, we repeatedly need to compute gradients and Hessians. Let $f : \mathbb{R}^n \mapsto \mathbb{R}$. Then its gradient at x, $\nabla f(x) : \mathbb{R}^n \mapsto \mathbb{R}^n$, is defined by

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

Note that the linear function

$$f(x) = a^T x = \sum_{i=1}^{n} a_i x_i$$

has gradient

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = a$$

and the quadratic function

$$f(x) = x^T Qx = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j$$

has gradient

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 2q_{11}x_1 + (q_{12}+q_{21})x_2 + (q_{13}+q_{31})x_3 + \cdots + (q_{1n}+q_{n1})x_n \\ (q_{12}+q_{21})x_1 + 2q_{22}x_2 + (q_{23}+q_{32})x_3 + \cdots + (q_{2n}+q_{n2})x_n \\ \vdots \\ (q_{1n}+q_{n1})x_1 + (q_{2n}+q_{n2})x_2 + (q_{3n}+q_{n3})x_3 + \cdots + 2q_{nn}x_n \end{bmatrix}$$

So if Q is symmetric, *i.e.* $q_{ij} = q_{ji}$ then we find that

$$\nabla f(x) = 2Qx$$

■ **Example B.3** The quadratic function

$$f(x) = 4x_1^2 + 4x_2^2 - 4x_1x_2$$

has gradient

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 8x_1 - 4x_2 \\ 8x_2 - 4x_1 \end{bmatrix}$$

The function can be represented as $f(x) = x^T Qx$ with

$$Q = \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}$$

According to our formulas,

$$\nabla f(x) = 2Qx = \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8x_1 - 4x_2 \\ -4x_1 + 8x_2 \end{bmatrix}$$

which agrees with our direct calculations. ■

■ **Example B.4** Combining the results above, we note that the quadratic function

$$f(x) = x^T Px + 2q^T x + r$$

has gradient

$$\nabla f(x) = 2Px + 2q.$$

■

B.2 Positive definite matrices and functions

Definition B.2.1 — Positive definite functions. We say that the function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is *positive semidefinite* if

$$f(x) \geq 0 \qquad \forall x \in \mathbb{R}^n$$

and that f is *positive definite* if

$$f(x) > 0 \qquad \forall x \in \mathbb{R}^n \text{ with } x \neq 0$$

Since a quadratic form $f(x) = x^T Qx$ is characterized by the matrix Q, it is natural to link positive definiteness of the quadratic form to properties of Q. This leads to the following definition.

Definition B.2.2 A square symmetric matrix $Q = Q^T \in \mathbb{R}^{n\times n}$ is called *positive semidefinite* if $f(x) = x^T Qx$ is a positive semidefinite function; it is called *positive definite* if $f(x) = x^T Qx$ defines a positive definite function.

We use the notation $Q \succeq 0$ to denote that Q is positive semidefinite, and $Q \succ 0$ to denote that it is positive definite. The following result will be useful.

Theorem B.2.1 The square symmetric matrix $Q = Q^T \in \mathbb{R}^{n\times n}$ is positive semidefinite if and only if all its eigenvalues are non-negative and real. A positive semidefinite matrix admits a representation $Q = R^T R$ where $R \in \mathbb{R}^{n\times n}$.

The square symmetric $Q = Q^T \in \mathbb{R}^{n\times n}$ is positive definite if and only if all its eigenvalues are positive and real. A positive definite matrix admits a representation $Q = R^T R$ where $R \in \mathbb{R}^{n\times n}$ is of full rank. A positive definite matrix is invertible (and its inverse is itself positive definite).

This theorem tells us that a positive definite quadratic form $x^T Qx$ can be written as $\sum_{i=1}^n z_i^2 = \sum_{i=1}^n (r_i^T x)^2$ for some vectors $r_i \in \mathbb{R}^n$. The quadratic form is zero for x which satisfy $r_i^T x = 0$ for all i; in a positive definite quadratic form, the matrix R spans $\mathbb{R}^n$ and we can only have $Rx = 0$ for $x = 0$. The next example illustrates these facts.

■ **Example B.5** The quadratic form $f(x) = x^T Qx$ with

$$Q = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

is positive semidefinite since Q has eigenvalues 2 and 0. We can express $x^T Qx$ as $x^T R^T Rx$ with

$$R = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

which corresponds to re-writing $f(x) = x_1^2 - 2x_1x_2 + x_2^2 = (x_1 - x_2)^2$. Note that $f(x) = 0$ for $x_1 = x_2$.

Similarly $f(x) = x^T Qx$ with

$$Q = \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}$$

is positive definite, since the eigenvalues of Q are 2 and 6. We can write $Q = R^T R$ with

$$R = \begin{bmatrix} 2 & -1 \\ 0 & \sqrt{3} \end{bmatrix}$$

which corresponds to expressing $f(x) = 4x_1^2 + 4x_2^2 - 4x_1x_2$ as $(2x_1 - x_2)^2 + 3x_2^2$. Note that $2x_1 - x_2$ and x_2 can only be simultaneously zero if $x_1 = x_2 = 0$. ■

Positive definite matrices and positive definite functions of complex arguments

The definitions of positive definite matrices and functions extend to complex matrices and vectors. In particular, we say that a complex-valued matrix $M \in \mathbb{C}^{n\times n}$ is positive definite, if and only if $z^* Mz$ is real and positive for all non-zero complex column vectors $z \in \mathbb{C}^n$. In these expressions, z^* is the complex conjugate transpose of z. We refer to, *e.g.* [10] for further deatils.

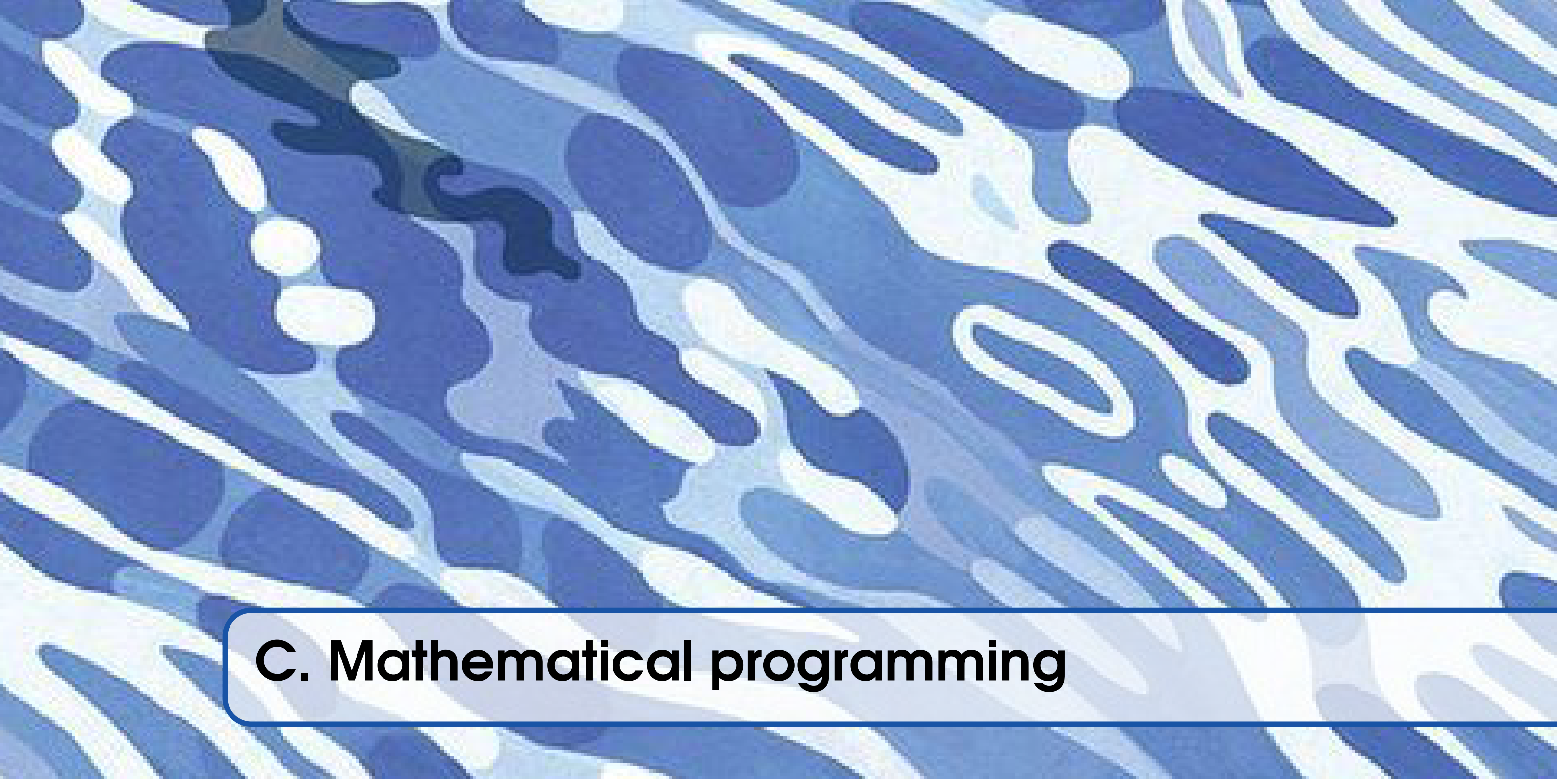
In these notes, we will only use the fact that if $Q \in \mathbb{R}^{n\times n}$ is positive definite, then

$$z^* Qz > 0$$

for all $z \in \mathbb{C}^n$ with $z \neq 0$. This follows by writing $z = x + iy$ with $x, y \in \mathbb{R}^n$ and evaluating

$$z^* Qz = (x^T - iy^T)Q(x + iy) = x^T Qx + y^T Qy.$$

The right-hand side is positive unless $x = y = 0$, which proves our claim.



C. Mathematical programming

This section contains a tutorial introduction to selected results from mathematical programming and convex optimization. They are useful for the developments in these notes, but by no means complete. We refer to the literature, such as [4, 6], for details.

C.1 Optimality conditions for unconstrained optimization

Consider the unconstrained optimization problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) \tag{C.1}$$

where $f:\mathbb{R}^n \mapsto \mathbb{R}$ is the *objective function* and $x\in\mathbb{R}^n$ is the *decision vector*. If f is continuously differentiable, then any minimizer $x^\star$ must satisfy the *first-order optimality condition*

$$\nabla f(x^\star) = 0. \tag{C.2}$$

(otherwise, we could improve the objective by adjusting $x^\star$ in the direction of the negative gradient). However, this condition is not sufficient for guaranteeing optimality. A vector x for which $\nabla f(x)=0$ could be a minimum, a maximum or a saddle point; see Figure C.1. In order to say more, we must put additional restrictions on the objective function. In these notes we will therefore limit the objective functions and constraints to be convex, as defined next.

C.2 Convexity and optimization

We begin by the definition of convex sets.

Definition C.2.1 The set $X\subseteq\mathbb{R}^m$ is *convex* if for any $x_1, x_2 \in X$ and every $\theta\in[0,1]$ we have $\theta x_1 + (1-\theta)x_2 \in X$.

In words, this definition states that a set X is convex if the line segment between any two points in X also lies in X; see Figure C.2.

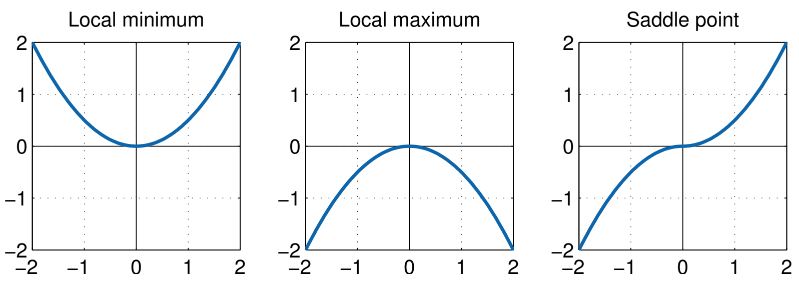


Figure C.1: Three continuously differentiable functions, whose gradients vanish at the origin. In the different functions, the origin is a local minima, local maxima and a saddle points, respectively.

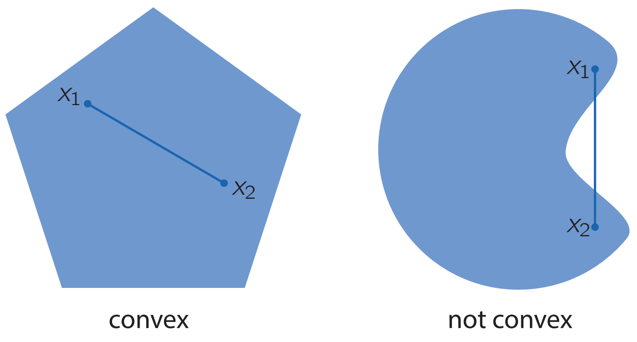


Figure C.2: A set is convex if every line segment between points in the set also lies in the set.

■ **Example C.1** One of the most basic convex sets is the half-space, defined by the solution set of a linear inequality

$$X = \left\{x \mid a^T x \leq b\right\}.$$

To prove that it is convex, we use the definition and consider $y = \theta x_1 + (1-\theta)x_2$ for $x_1, x_2 \in X$. Then, for $\theta \in [0,1]$ it holds that

$$a^T y = \theta a^T x_1 + (1-\theta)a^T x_2 \leq \theta b + (1-\theta)b = b$$

where the inequality follows since x_1 and x_2 both lie in X (that is, satisfy $a^T x_1 \leq b$ and $a^T x_2 \leq b$, repsectively). Hence $y \in X$ for every value of $\theta \in [0,1]$, so X is convex. ■

Definition C.2.2 The function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is *convex* if its domain is a convex set and it holds that for all $x, y \in \operatorname{dom} f$ and for every $\theta \in [0,1]$ we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y) \tag{C.3}$$

The geometrical interpretation of (C.3) is that the line segment between $(x, f(x))$ and $(y, f(y))$ always lies above the graph of f; see Figure C.3.

■ **Example C.2** By a similar calculation as in Example C.1, we can easily show that affine functions

$$f(x) = a^T x + b$$

are convex. We will now show that quadratic functions

$$f(x) = x^T P x$$

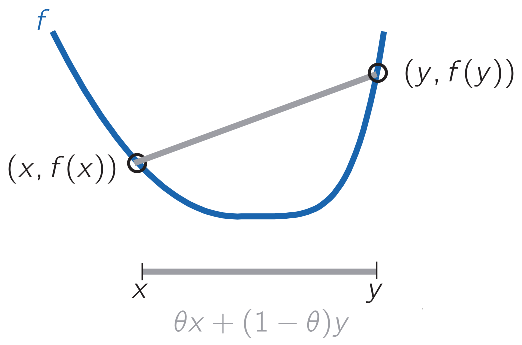


Figure C.3: A function f is convex if the line segment between $(x, f(x))$ and $(y, f(y))$ lies above the graph of f for every x and y in its domain.

are convex, provided that P is a positive semidefinite matrix. We then have that

$$\begin{aligned}
&f(\theta x + (1-\theta)y) - \theta f(x) - (1-\theta)f(y) = \\
&= \theta^2 x^T P x + (1-\theta)^2 x^T P y - 2\theta(1-\theta)x^T P y - \theta x^T P x - (1-\theta) y^T P y = \\
&= \theta(\theta-1)\left[(x-y)^T P (x-y)\right] \leq 0,
\end{aligned}$$

where the last inequality follows since $\theta(\theta-1) < 0$ and P is positive semidefinite.

Finally, we also notice that the sum of two convex functions is convex (just apply the definition), so the more general quadratic form

$$f(x) = \frac{1}{2}x^T P x + q^T x + r$$

is convex if P is positive semidefinite. ■

If f is continuously differentiable, then we can give an alternative characterization of convex functions which is often easier to work with.

Proposition C.2.1 A continuously differentiable function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if and only if

$$f(y) \geq f(x) + (y-x)^T \nabla f(x) \qquad \text{for all } x, y \in \text{dom } f \tag{C.4}$$

Condition (C.4) implies that for differentiable convex functions, the linearization at any point is a global lower bound on the function; see Figure C.4. A number of important properties of convex functions follow immediately from Definition C.2.1: stationary points (x for which the gradient vanishes) of convex functions are necessarily minima; any local minimum is also a global minimum; and the first-order optimality condition (C.2) is both necessary and sufficient.

C.3 Constrained convex optimization problems

In practice, the decision vector often has to satisfy additional constraints. We will thererfore study constrained optimization problems on the form

$$\begin{array}{lll}
\text{minimize} & f_0(x) & \\
\text{subject to} & f_i(x) \leq 0 & i = 1, \ldots, m \\
 & g_i^T x = h_i & i = 1, \ldots, p
\end{array} \tag{C.5}$$

In this formulation, $f_0(x)$ is the *objective function* representing the operating cost of the system while $x \in \mathbb{R}^n$ is the *decision vector* containing the free variables. In addition $f_i(x) \leq 0$ describe

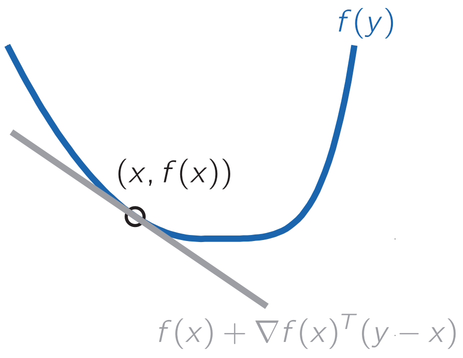


Figure C.4: For continuously differentiable convex functions, every linearization is a global lower bound.

inequality constraints and $g_i^T x = h_i$ describe *linear equality constraints*. We say that x is *feasible* if it satisfies all constraints. The optimization problem (C.5) is *feasible* if it admits at least one feasible x. A vector $x^\star$ is said to be *optimal* if it attains the smallest value of f_0 among all feasible x. We let $p^\star = f_0(x^\star)$ denote the *optimal value* of (C.5).

It is sometimes convenient to write the optimization problem (C.5) on the compact form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & x \in X \end{array}$$

where we have introduced the *feasible set*

$$X = \left\{ x \mid f_i(x) \leq 0,\ i = 1, \ldots, m,\ \wedge g_i^T x - h_i = 0,\ i = 1, \ldots, p \right\}. \tag{C.6}$$

as the admissible decision vectors which satisfy all constraints.

We will focus on *convex optimization problems* in which f_0 is a convex function and X is a convex set. A simple calculation reveals that X defined in (C.6) is a convex set when all inequality constraint functions f_i are convex and all equality constraint functions are affine. Convex optimization problems admit a powerful and elegant theory, and can be solved to global optimality using a range of efficient numerical solvers; see *e.g.* [6] for more details.

The two most well-known classes of convex programming problems are linear and quadratic programs. In a *linear programming (LP) problem* one minimizes a linear function subject to linear inequality and equality constraints:

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & a_i^T x \leq b_i & i = 1, \ldots, m \\ & g_i^T x = h_i & i = 1, \ldots, p \end{array} \tag{C.7}$$

A *quadratic programming (QP) problem*, on the other hand, considers the minimization of a convex quadratic function subject to linear constraints:

$$\begin{array}{lll} \text{minimize} & x^T P x + 2q^T x + r & \\ \text{subject to} & a_i^T x \leq b_i & i = 1, \ldots, m \\ & g_i^T x = h_i & i = 1, \ldots, p \end{array} \tag{C.8}$$

Both linear and quadratic programming problems have been studied extensively and admit a rich and useful theory. In addition, the numerical solvers for these problems have reached a high level of maturity and can routinely solve problems with millions of variables and constraints.

C.4 Optimality conditions for constrained convex optimization

For unconstrained convex optimization problems, the first-order optimality condition allows for a simple analytical characterization of the optimizers. The corresponding condition for constrained convex optimization problems reads that $x^\star$ must satisfy

$$\langle \nabla f(x^\star), y - x^\star \rangle \geq 0 \qquad \forall y \in X.$$

This condition is unfortunately much more difficult to check than the unconstrained counterpart, and has therefore limited practical use.

A more convenient technique for studying constrained convex optimization is the *method of Lagrange multipliers*. Given a problem on the form (C.5), one constructs an associated *Lagrangian function* $L : \mathbb{R}^n \times \mathbb{R}^m_+ \times \mathbb{R}^p \mapsto \mathbb{R}$

$$L(x,\lambda,\mu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \mu_i (g_i^T x - h_i).$$

Here $\lambda_i \geq 0$ is a *Lagrange multiplier* for the inequality constraint $f_i(x) \leq 0$, while μ_i is a Lagrange multiplier for the equality constraint $g_i^T x - h_i = 0$. Note that if x is feasible (*i.e.*, satisfies the constraints), then

$$L(x,\lambda,\mu) \leq f_0(x)$$

and that equality holds if $\lambda_i f_i(x) = 0$ for all i. To explore this lower bound further, one introduces the *dual function* $g : \mathbb{R}^m_+ \times \mathbb{R}^p \mapsto \mathbb{R}$ as

$$g(\lambda,\mu) = \inf_{x \in \mathcal{D}} L(x,\lambda,\mu) = \inf_{x \in \mathcal{D}} \left\{ f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \mu_i^\star (g_i^T x - h_i) \right\}$$

Here, $\mathcal{D} = \cap_{i=0}^{m} \operatorname{dom} f_i$, which we will assume to be non-empty. A key property of the dual function is that it lower bounds the optimal value of (C.5).

Proposition C.4.1 If $\lambda_i \geq 0$ for $i = 1, \dots, m$, then

$$g(\lambda,\mu) \leq p^\star$$

Note that g could be unbounded (*i.e.* $-\infty$). To restrict the discussion to non-trivial lower bounds, one sometimes introduces the *dual feasible set*

$$\{(\lambda,\mu) \mid \lambda \succeq 0,\ (\lambda,\mu) \in \operatorname{dom}(g)\}.$$

Since g is a lower bound to $p^\star$, it is natural to try to make this bound as tight as possible, *i.e.* to

$$\begin{array}{ll} \text{maximize} & g(\lambda,\mu) \\ \text{subject to} & \lambda \succeq 0 \end{array}$$

This optimization problem is called the *dual problem* to (C.5), and we will denote its optimal value by $d^\star$. The next result follows immediately from our discussion.

Proposition C.4.2 Weak duality, *i.e.* $d^\star \leq p^\star$ always holds.

More importantly, under several mild conditions, convex optimization problems allow us to guarantee *strong duality*, *i.e.* that $d^\star = p^\star$. One such condition, known as *Slater's condition*, is exploited in the next result:

Theorem C.4.3 If $f_i(x)$, $i = 0, \dots, m$ are convex and there exists an $x \in \operatorname{int} \mathcal{D}$:

$$f_i(x) < 0, \quad i = 1, \dots, m \text{ and } g_i^T x - h_i = 0, \quad i = 1, \dots, p$$

then $d^\star = p^\star$.

Under strong duality, we can derive several additional useful results. To this end, let $x^\star$ be primal optimal and $(\lambda^\star, \mu^\star)$ be a dual optimal point. Then, if strong duality holds, we have

$$f_0(x^\star) = g(\lambda^\star, \mu^\star) = \inf_{x \in \mathcal{D}} \left\{ f_0(x) + \sum_{i=1}^{m} \lambda_i^\star f_i(x) + \sum_{i=1}^{p} \mu_i^\star (g_i^T x - h_i) \right\} \leq$$
$$\leq f_0(x^\star) + \sum_{i=1}^{m} \lambda_i^\star f_i(x^\star) + \sum_{i=1}^{p} \mu_i^\star (g_i^T x^\star - h_i) \leq f_0(x^\star).$$

Since all inequalities must hold with equality, we conclude that $x^\star$ minimizes $L(x, \lambda^\star, \mu^\star)$ and that the so-called *complementary slackness* $\lambda_i^\star f_i(x^\star) = 0$ must hold for all $i = 1, \dots, m$. This leads to the *Karush-Kuhn-Tucker (KKT) conditions* for optimality:

$$\begin{aligned} \nabla f_0(x^\star) + \sum_{i=1}^{m} \lambda_i^\star \nabla f_i(x^\star) + \sum_{i=1}^{p} \mu_i^\star g_i &= 0 & & \\ \lambda_i^\star f_i(x^\star) &= 0 & & i = 1, \dots, m \\ f_i(x^\star) &\leq 0 & & i = 1, \dots, m \\ g_i^T x^\star - h_i &= 0 & & i = 1, \dots, p \\ \lambda_i^\star &\geq 0 & & i = 1, \dots, m. \end{aligned}$$

Here, the first equality is the first-order optimality condition for $x^\star$ minimizing $L(x, \lambda^\star, \mu^\star)$, the second set of equations describe complementary slackness, the third and fourth set of equations are primal feasibility and the final set of conditions describe dual feasibility.

Theorem C.4.4 Consider the constrained optimization problem (C.5) under the assumption that $f_0, \dots, f_m$ are continuously differentiable and convex. Then, if Slater's condition holds, the KKT conditions are necessary and sufficient for optimality.

The next example illustrates a typical application of the method of Lagrange multipliers.

■ **Example C.3** To solve the constrained optimization problem

$$\begin{array}{ll} \text{minimize} & x^T x \\ \text{subject to} & Cx = d \end{array}$$

using the method of Lagrange multipliers, we first form the Lagrangian function

$$L(x, \mu) = x^T x + \mu^T (Cx - d)$$

The associated dual function is

$$g(\mu) = \inf_x L(x, \mu).$$

By the first-order optimality conditions, the minimizing x is $x = -C^T \mu / 2$ so

$$g(\mu) = L(-\frac{1}{2} C^T \mu, \mu) = -\frac{1}{4} \mu^T C C^T \mu - \mu^T d.$$

The optimal dual value

$$d^\star = \sup_\mu g(\mu)$$

can also be computed by application of the first-order optimality conditions. Doing so, we find that the optimal Lagrange multiplier must satisfy

$$-\frac{1}{2}CC^T\mu^\star - d = 0$$

Under the assumption that CC^T is invertible, $\mu^\star = -2(CC^T)^{-1}d$ and

$$d^\star = d^T(CC^T)^{-1}d.$$

Since x^Tx is convex, if there is an x such that $Cx = d$, then by Theorem C.4.3

$$p^\star = d^\star = d^T(CC^T)^{-1}d$$

and an optimal solution is given by the minimizer of $L(x, \mu^\star)$, *i.e.*

$$x^\star = C^T(CC^T)^{-1}d.$$

■

Bibliography

[1] S. Abbott. *Understanding Analysis*. Springer Verlag, 2015 (cited on page 32).

[2] B. D. O. Anderson and J. B. Moore. *Linear optimal control*. 1971 (cited on pages 68, 80).

[3] K. J. Åström and B. Wittenmark. *Computer controlled systems: theory and design*. 2nd. Prentice Hall, 1990 (cited on pages 21, 23).

[4] D. Bertsekas. *Nonlinear programming*. 2nd. Athena Scientific, 1999 (cited on page 125).

[5] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming*. Athena Scientific, 2012.

[6] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787 (cited on pages 125, 128).

[7] Gene Franklin, J. D. Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems (5th Edition)*. 5th edition. Prentice Hall, 2005 (cited on page 21).

[8] E. G. Gilbert and K. T. Tan. “Linear systems with state and control constraints: the theory and application of maximal output admissible sets”. In: *IEEE Transactions on Automatic Control* 36.9 (Sept. 1991), pages 1008–1020 (cited on page 38).

[9] M. Grundelius. “Methods for Control of Liquid Slosh”. eng. Lund University, 2001, page 167 (cited on page 51).

[10] R. A. Horn and C. R. Johnson, editors. *Matrix Analysis*. New York, NY, USA: Cambridge University Press, 1986. ISBN: 0-521-30586-1 (cited on page 124).

[11] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. Chichester: Wiley-Interscience, 1972 (cited on page 71).

[12] J. Lofberg. “YALMIP : a toolbox for modeling and optimization in MATLAB”. In: *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*. Sept. 2004, pages 284–289. DOI: `10.1109/CACSD.2004.1393890` (cited on page 48).

[13] A. M. Lyapunov. “The General Problem of the Stability of Motion”. Russian. Doctoral dissertation. Univ. Kharkov, 1892 (cited on page 29).

[14] James Rawlings. *Model predictive control : theory and design*. Madison, Wis: Nob Hill Pub, 2009. ISBN: 0975937707 (cited on page 100).

[15] P. O. M. Scokaert and J. B. Rawlings. "Constrained linear quadratic regulation". In: *IEEE Transactions on Automatic Control* 43.8 (Aug. 1998), pages 1163–1169 (cited on page 100).

[16] P. C. Young and J. C. Willems. "An approach to the linear multivariable servomechanism problem†". In: *International Journal of Control* 15.5 (1972), pages 961–979 (cited on page 74).