Applied Mathematics Street Fighting

Quill Healey

June 12, 2024

Contents

Chapter 1

Introduction

1.1 What are These Notes?

[BV04]

Part I

Theory

Chapter 2

Linear Algebra

to adds:

- incidence matrix page 132 VMLS (networks there too)
- bidiagonal matrix page 312 [BV04]

2.1 Eigenvalues and Eigenvectors

2.1.1 Stability

Continuous Time Systems

Discrete Time Systems

[PG24] **HW0 Q1**. *Discrete-time LTI stability.* Consider the system $x_{t+1} = Ax_t + Bu_t$, where

$$A = \begin{bmatrix} 4/5 & 0 & 0 \\ 0 & \sqrt{3} & 1 \\ 0 & -1 & \sqrt{3} \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

(a) Explain whether or not this sytem is "open-loop stable" (asymptotically stable for $u_t \equiv 0$).
Response. This sytem is unstable. To see this, note $|\lambda_1| = |\lambda_2| \approx 2$. (Most likely they equal two, the numerical computation via Python yields a magnitude of 1.99... with 15 trailing 9s). Recall that for a discrete time LTI system to be open-loop stable, the magnitude of all eigenvalues must be less than one.
(b) Design a linear feedback controller $u_t = Kx_t$ with fixed gain matrix $K \in \mathbf{R}^{2\times 3}$ such that the closed-loop system is asymptotically stable.
Response. This is a **state feedback control** problem where K is the **state-feedback gain matrix**. In this setting, we can rewrite the LTI system as

$$x_{t+1} = Ax_t + Bu_t = Ax_t + B(Kx_t) = (A + BK)x_t, \quad t = 1, 2, \ldots$$

2.2 Linear Transformations

Chapter 3

Matrix Calculus

3.1 Notation

3.2 Rethinking the Derivative

3.3 Some Analysis

3.4 Some Geometry

3.5 Calculus Composition Rules

- be careful with composing differentials versus derivatives versus gradients
- Move right to "beyond high school calculus," most importantly now must be careful with commuting.
- We are assuming differentiability throughout.

Sum Rule

The generalized matrix calculus sum rule is exactly what you would think it would be. Given $f : \mathbf{R}^n \to \mathbf{R}^m$ where $f(x) = g(x) + h(x)$,

$$df = dg + dh,$$

which can be expanded (according to the definition of a differential) as

$$Df(x)dx = Dg(x)dx + Dh(x)dx = (Dg(x) + Dh(x))dx.$$

Therefore,

$$Df(x) = Dg(x) + Dh(x).$$

Product Rule

Consider $f : \mathbf{R}^n \to \mathbf{R}^m$ defined as $f(x) = g(x)h(x)$. The product rule derivation is as follows.

$$df = f(x + dx) - f(x) = g(x + dx)h(x + dx) - g(x)h(x).$$

Using that $g(x+dx) = g(x)+Dg(x)dx$ and $h(x+dx) = h(x)+Dh(x)dx$, the differential df can be expanded as

$$\begin{aligned} df &= (g(x) + Dg(x)dx)\,(h(x) + Dh(x)dx) - g(x)h(x) \\ &= g(x)h(x) + g(x)Dh(x)dx + Dg(x)dxh(x) + Dg(x)dxDh(x)dx - g(x)h(x), \end{aligned}$$

ignoring the higher order terms, removing like terms, and being mindful *to not commute terms*, we are left with

$$df = g(x)Dh(x)dx + Dg(x)dxh(x).$$

This of course should look like the typical product rule seen for Calculus I derivatives, but we have mixed differentials and derivatives.

$$df = g(dh) + (dg)h \quad \text{and} \quad Df(x)dx = g(x)\,(Dh(x)dx) + (Dg(x)dx)\,h(x).$$

- Notice that we have not written that $Df(x) = g(x)Dh(x) + Dg(x)h(x)$.

Chain Rule

Scalar composed with vector value

3.6 Calculus Atoms

3.6.1 Vector Functions

Linear and Affine Functions

Take $f : \mathbf{R}^n \to \mathbf{R}^m$ defined as $f(x) = Ax - b$ for some $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$.

$$df = d(Ax - b) = A(x + dx) - b - (Ax - b) = Adx,$$

so

$$Df(x) = A \quad \text{and} \quad \nabla f(x) = A^T.$$

Euclidean Inner Product

Take $f : \mathbf{R}^n \to \mathbf{R}$ defined as $f(x) = x^T x$.

$$\begin{aligned} df = d(x^T x) &= (x + dx)^T (x + dx) - x^T x \\ &= x^T x + x^T dx + (dx)^T x + (dx)^2 - x^T x \\ &= 2x^T dx, \end{aligned}$$

where the last equality holds because a is always equal to a^T when $a \in \mathbf{R}$. Therefore,

$$Df(x) = 2x^T \quad \text{and} \quad \nabla f(x) = 2x.$$

Quadratic Form

Consider $f : \mathbf{R}^n \to \mathbf{R}$ defined as $f(x) = x^T Ax$ for some $A \in \mathbf{R}^{n\times n}$.

$$\begin{aligned} df = d(x^T Ax) &= (x + dx)^T A(x + dx) - x^T Ax \\ &= x^T Ax + x^T Adx + (dx)^T Ax + (dx)^T Adx - x^T Ax \\ &= x^T Adx + x^T A^T dx \\ &= x^T (A + A^T)dx, \end{aligned}$$

so

$$Df(x) = x^T (A + A^T) \quad \text{and} \quad \nabla f(x) = \left(x^T (A + A^T)\right)^T = (A + A^T)x.$$

(Note that $(A + A^T)^T = (A + A^T) \Leftrightarrow A + A^T \in \mathbf{S}^n$.)

Quadratic Form Restricted Case

Consider the same function $f : \mathbf{R}^n \to \mathbf{R}$ where $f(x) = x^T Ax$, but now $A \in \mathbf{S}^n$. The above differential derivation still holds:

$$df = x^T (A + A^T)dx,$$

but because $A = A^T$, we can further simplify the differential to

$$df = 2x^T Adx.$$

Consequently,

$$Df(x) = 2x^T A \quad \text{and} \quad \nabla f(x) = 2A^T x = 2Ax,$$

where we again use that $A^T = A$.

3.6.2 Matrix Functions

3.6.3 Summary

3.7 (Bonus) Automatic Differentiation

Chapter 4

Mathematical Optimization

4.1 Optimization Problems

4.2 Basic Optimality Conditions

4.3 Linear Optimization

4.3.1 Reformulations

Problems Involving ℓ_1- and ℓ_∞- norms

Just as one learns to see a sum of squares as the squared ℓ_2-norm of the vector whose entries are the expressions in the sum of squares being squared, for formulating LPs it is critical to develop the habit of seeing the sum of absolute values as the ℓ_1-norm and the maximum of absolute values as the ℓ_∞-norm. While non-trivial LPs have objective functions and constraint functions which can be harder to write in matrix-vector notation, these problems can be reformulated using the same techniques shown below for optimization problems which are written in matrix-vector notation. Furthermore, the following exercise proves the equivalence between basic norm optimization problem (originally written in matrix-vector notation) and their corresponding LPs. These results provide the mathematical grounding that guarantees we can rewrite more exotic mathematical optimization problems involving compositions of maximums, sums, and absolute values as LPs, and they also provide a roadmap for deriving LP reformulations ourselves. An applied reformulation example can be found on page 35.

The following exercise proves the equivalence between norm minimization problems and their corresponding LPs. However, because (in general) showing problem equivalence can be tricky, we'll use the simplicity of these LP equivalencies to work through two equivalence approaches for each subproblem. Firstly, we'll *derive* the LP by starting with its equivalent norm problem and then creating a *sequence* of equivalent problems until we arrive at the LP. This is a valid approach to showing equivalency. This is also a *first principles* approach, which can be useful for reformulating optimization problems as LPs when the original problem is harder to write in matrix-vector notation. Once we have the final LP, we'll then directly show its equivalence to the original norm problem. Being able to provide this type of detailed explanation for the relation between two problem's optimal solutions is also a vital skill and has its own stylistic approach.

[BV04] **Exercise 4.11**. Formulate the following problems as LPs. Explain in detail the relation between the optimal solution of each problem and the solution of its equivalent LP.
(a) Minimize $\|Ax - b\|_\infty$ (ℓ_∞-norm approximation).

Response.
(Approach 1). Our initial problem is the unconstrained problem

$$\text{minimize } \|Ax-b\|_\infty = \max_k \left\{ \left|a_k^T x - b_k\right| \right\}. \tag{$\mathcal{P}_1$}$$

We can rewrite this problem in *epigraph form*,

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \max_k \left\{ \left|a_k^T x - b_k\right| \right\} \leq t, \end{array} \tag{$\mathcal{P}_2$}$$

which is always an equivalent transformation. Next, we use that the single constraint

$$\max_k \left\{ \left|a_k^T x - b_k\right| \right\} \leq t$$

implies that each of the expressions

$$\left|a_k^T x - b_k\right|, \quad k = 1, \dots, m$$

must also be less than or equal to t, *i.e.*, we have the third problem

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \left|a_k^T x - b_k\right| \leq t, \quad k = 1, \dots, m. \end{array} \tag{$\mathcal{P}_3$}$$

Finally, using the definition of the absolute value (which I've found very useful to keep at the front of one's mind when working on these problems), $|a| = \max\{a, -a\}$, we have that the constraints in ($\mathcal{P}_3$) can be rewritten as

$$a_k^T x - b_k \leq t, \; -(a_k^T x - b_k) \leq t, \quad k = 1, \dots m,$$

since if the maximum of $a_k^T x - b_k$ and $b_k - a_k^T x$ is less than or equal to t then both the terms must be. We therefore arrive at the following LP

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & -t\mathbf{1} \preceq Ax - b \preceq t\mathbf{1}, \end{array} \tag{$\mathcal{P}_4$}$$

which we've written more compactly in matrix-vector notation where $\mathbf{1} \in \mathbf{R}^m$. We also know that this LP is equivalent to the original problem because every problem transformation we performed is reversible, *i.e.*, we have equivalency through the following problem transformation sequence

$$\mathcal{P}_1 \iff \mathcal{P}_2 \iff \mathcal{P}_3 \iff \mathcal{P}_4.$$

(Approach 2). We want to show that the optimal solution of

$$\text{minimize } \|Ax - b\|_\infty$$

is also an optimal solution of

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & -t\mathbf{1} \preceq Ax - b \preceq t\mathbf{1}, \end{array}$$

and vice-versa. To see this, we first need to address that the LP has the additional optimization variable, t. Because we wish to show that optimizing the two problems *over* x produces the same solution, it's practical to consider the optimal value of the LP as a function of x, *i.e.*, consider the LP and *optimize over* t, with corresponding optimal cost $p^*(x)$. Using the same transformations as performed above, the constraints in the LP can be written as

$$\left|a_k^T x - b_k\right| \le t, \quad k = 1, \ldots, m.$$

Obviously, if x is fixed and we are minimizing over t, the optimal value of the LP is $\max_k \left\{\left|a_k^T x - b_k\right|\right\}$, since this is the greatest lower bound that t is constrained to. In other words, $p^*(x) = \|Ax - b\|_\infty$. Therefore, optimizing over t and x simultaneously is equivalent to the original problem.

(b) Minimize $\|Ax - b\|_1$ (ℓ_1-norm approximation).

Response.

(Approach 1). Consider the unconstrained problem

$$\text{minimize } \|Ax - b\|_1 = \sum_{i=1}^{m} \left|a_i^T x - b_i\right|,$$

which can be written in the equivalent epigraph form

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \sum_{i=1}^m \left|a_i^T x - b_i\right| \le t. \end{array}$$

Now, introduce a new optimization variable $s \in \mathbf{R}^m$, which is to be optimized over along with x and t, and consider

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & \mathbf{1}^T s \le t \\ & \left|a_i^T x - b_i\right| \le s_i, \quad i = 1, \ldots m. \end{array}$$

Fixing x, the previous two problems can be seen as equivalent by imagining t pushing the sum $\mathbf{1}^T s$ to be as small as possible, and because all s_i entries are **uncoupled** from one another, minimizing t will push each s_i to its lower bound $\left|a_i^T x - b_i\right|$.

From here, reaching an LP requires the same transformations as performed in (a), so we will not perform them. However, it is worth noting the constraint

$$\mathbf{1}^T s \le t,$$

which consists entirely of *auxilary* optimization variables. Since the objective of the most recent optimization problem is to minimize t, clearly this is equivalent to minimizing $\mathbf{1}^T s$. We therefore have found our final form equivalent LP

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s \\ \text{subject to} & -s \preceq Ax - b \preceq s. \end{array}$$

(Approach 2). We will show that the solution to

$$\text{minimize } \|Ax - b\|_1 = \sum_{i=1}^{m} \left| a_i^T x - b_i \right|,$$

is equivalent to

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s \\ \text{subject to} & -s \preceq Ax - b \preceq s, \end{array}$$

and vice-versa. The most important observation

Chapter 5

Duality

5.1 Conjugate Function

Recall that the conjugate function

$$f^*(y) = \sup_{x \in \mathbf{dom}\, f} \left(y^T x - f(x) \right)$$

5.1.1 Examples (Conjugate Atoms)

Chapter 6

Probability Theory

About:

- chapter will examine probability theory through the lense of convex optimization. Probability theory itself is not the main focus.

Todos

- Where to add chebychev
- simple chebychev example page 54 VMLS
- correlation coefficient page 60 VMLS
-

Outline

- Basics (what you need probabilistically to proceed) subsection 1: the probability theory. subsection 2: Viewing the probabilistic objects through the lense of convex optimization
- Sets: Exercise 2.15
- Functions: Exercise 3.24.
- Basic Problems

6.1 Basics

6.1.1 Probabilistics Objects and Notation

I will adopt the notation used by the source material.

- Unlike in statistical texts, random variables (RVs) will not by default be a capital letter. That is, never assume that a capital lettered variable (or any other variable, for that matter) is a random variable, even in a statistical context. If x or X is a random variable, it will be declared as such.

- A random variable, x, may be real valued or it may be vector valued. That is to say, the notation for a random variable or a *random vector* will amost always be the same. However, it will always be specified upon instantiation (using computer science/software engineering dialect) whether x is a random variable or whether it is a random vector.

- In this chapter, we will primarily work with random variables which takes on discrete values. That is, $x \sim (\Omega_x, E_x, \mathbf{Prob}_x)$ where $\Omega_x = \{a_1, a_2, \ldots, a_n\} \subset \mathbf{R}^n$ and it is assumed that $a_1 < a_2 < \cdots < a_n$.

- The fundamental probability space $(\Omega_\omega, E_\omega, \mathbf{Prob}_\omega)$ which induces the random variable's probability space via the relation $x : \Omega_\omega \to \Omega_x$ is irrelevant to our analysis.

- The probability of an outcome a_i in Ω_x ocurring will be denoted in the following ways:

$$p_x(a_i) = \mathbf{Prob}_x\left(a_i\right) = \mathbf{Prob}\left(x = a_i\right) = p_i.$$

  We'll refer to $p_x(\cdot)$ as the *probability mass function* (PMF) for the random variable x and to $\{p_i\}_{i=1}^n$ as the RV's distribution.

- We'll let $F_x(\alpha) = \mathbf{Prob}\left(x \leq \alpha\right)$ be the *cumulative distribution function* (CDF) for the random variable x.

6.1.2 Discrete Probabilistic Objects through a Convex Lens

Before proceeding to exercises and, it is helpful to collect the above objects

Before proceeding to exercises it is helpful to collect some objects. We take x to be a discrete random variable on a finite sample space, as defined above.

- The CDF of x can be expanded as

$$F_x(\alpha) = \mathbf{Prob}\left(x \leq \alpha\right) = \mathbf{Prob}\left(x \leq \alpha_k\right) = \sum_{i=1}^{k} p_i,$$

  where $k = \max\{j \mid a_j \leq \alpha\}$. It is **critical** to note that k is a fixed integer, **independent of** p. Furthermore, the CDF is just a linear function of p.

- The *expectation* (or weighted average) of x, $\mathbf{E}\, x = \sum_{i=1}^n p_i a_i = p^T a$, is a linear function of p.

- When we transform a random variable, $x \to f(x)$, the expectation becomes

$$\mathbf{E}\, f(x) = \sum_{i=1}^{n} p_i f(a_i),$$

which **is still a linear function of** p. (A curious reader may wonder what types of transformations are valid and how this works. Since probability theory is not the focus of these notes, accept the above statement as true since it will be true for all examples and exercises in these notes. However, to answer this question I recommend Murphy and Lay).

- The *variance* of x, which can be thought of as a measure of how much x typically deviates from its expectation, is defined as $\mathbf{Var}\, x = \mathbf{E}\left[(x - \mathbf{E}\, x)^2\right]$. However, we can rewrite the variance using common properties of the expectation operator as

$$\begin{aligned}\mathbf{Var}\, x &= \mathbf{E}\left[(x - \mathbf{E}\, x)^2\right] \\ &= \mathbf{E}\left[x^2 - 2x\mathbf{E}\, x + (\mathbf{E}\, x)^2\right] \\ &= \mathbf{E}\, x^2 - 2(\mathbf{E}\, x)(\mathbf{E}\, x) + (\mathbf{E}\, x)^2 = \mathbf{E}\, x^2 - (\mathbf{E}\, x)^2.\end{aligned}$$

To remember this more tractable expression, firstly note that the variance of a random variable is nonnegative. This can be seen mathematically from the definition of **Var**, and it should also be intuitive when thinking of the variance *as a measure*. Furthermore, recall the general form of Jensen's inequality: $f(\mathbf{E}\, x) \leq \mathbf{E}\, f(x)$, where f is a convex function. Moving the left hand side to the right hand side and using $x \xrightarrow{f} x^2$ yields our desired expression.

- The quartile of x

6.2 Sets

With these objects at our disposal, most of the solutions to problems in **Exercise 2.15** are immediately apparent. Furthermore, we just address the following.

(a-e) Note that each operator on x is linear in p. Furthermmore, the mixing of linear expressions in p with inequalities ($\leq$ or $\geq$) does not result in nonconvex conditions on P.

(f-g) Firstly, let's utilize Chapter Three composition techniques:

  - $\mathbf{E}\, x^2$, as already established, is linear in p.
  - $f : \mathbf{R} \to \mathbf{R}$ defined as $f(x) = x^2$ is convex. When $g : \mathbf{R} \to \mathbf{R}$ is also convex, $f(g(x)) = (g(x))^2$ is convex. Furthermore, $(\mathbf{E}\, x)^2$ is a convex quadratic in p.
  - The expression (*linear function* $-$ *convex quadratic function*) is a *concave quadratic expression.*

Using $f(p) = \mathbf{Var}\, x = \mathbf{E}\, x^2 - (\mathbf{E}\, x)^2$ to emphasize that the variance is a concave quadratic function of p, (f) and (g) follow from the recollection that $\{p \in P \mid f(p) \geq \alpha\}$ is a convex set while $\{p \in P \mid f(p) \leq \alpha\}$ is not. Additonally, note that the expression in the answer key for $(\mathbf{E}\, x)^2$ can be derived as follows:

$$(\mathbf{E}\, x)^2 = (p^T a)^2 = (p^T a)(p^T a) = (p^T a)(p^T a)^T = (p^T a)(a^T p) = p^T A p.$$

(h) The picture in the solutions posted to EE364a's Stanford Engineering Everywhere page is very useful.

Part II

Applications Part I

Chapter 7

Approximation and Fitting

First use that

$$\text{minimize } f_0(x) = \|Ax - b\|_2$$

is equivalent to (since f_0 is nonnegative)

$$\text{minimize } f_0(x)^2 = \|Ax - b\|_2^2\,,$$

where $f_0(x) = \|Ax - b\|_2^2 = (Ax - b)^T (Ax - b)$. Important observations

- f_0 is convex.
- The problem is unconstrained.

Furthermore, we know that the globally optimal solution, x^*, satisfies $\nabla f_0(x^*) = 0$. We use this as an opportunity to practice matrix calculus. Specifically, we will use differentials to derive ∇f_0. Start by writing out

$$df_0 = d\Big((Ax - b)^T (Ax - b)\Big) = (A(x + dx) - b)^T (A(x + dx) - b) - (Ax - b)^T (Ax - b)\,.$$

Recall that we wish to transform this equation into the form $df_0 = Df_0(x)dx$. Or rather, we want to re-arrange the expression for df_0 so that the dx terms are collected in one place and there's a clear expression for $Df_0(x)$. (Remember that depending on the complexity of the function, it's not always the case that the dxs will be collected on the right, as implied above.) Of course, because we've already developed matrix calculus atoms and compositions, we can take a shortcut. Rewrite $f_0(x) = g(h(x))$ where $g : \mathbf{R}^m \to \mathbf{R}$ and $h : \mathbf{R}^n \to \mathbf{R}^m$ are defined as

$$g(x) = x^T x \quad \text{and} \quad h(x) = Ax - b,$$

with corresponding matrix calculus atoms

$$Dg(x) = 2x^T \quad \text{and} \quad Dh(x) = A.$$

Also observe what this implies about f_0. Instead of being a mapping from n-dimensional Euclidean space to the real line, it is actually the composition mapping $f_0 : \mathbf{R}^n \to \mathbf{R}^m \to \mathbf{R}$. This isn't profound, but it can be helpful to keep in mind when performing a derivation such as this one since it reinforces the danger of absentmindedly commuting terms. Employing the chain rule, $Df_0(x) = Dg(h(x)Dh(x))$,

$$Df_0(x) = 2(Ax - b)^T A, \quad \text{so} \quad \nabla f_0(x) = \left(2(Ax - b)^T A\right)^T = 2A^T(Ax - b).$$

Equating $\nabla f_0(x) = 0$, we arrive at the normal equations

$$A^T Ax = A^T b,$$

with associated solution

$$x^* = (A^T A)^{-1} A^T b.$$

7.1 Forecasting

Sources:

- AR Model (page 28 VMLS)
- RMS Prediction Error (page 50 VMLS)
- Dirichlet energy page 66 VMLS
- Finance example page 67 VMLS
- Time series auto-correlation page 67
- back-test timing page 127
- Time series smoothing page 138 VMLS (convolution)
- Markov model page 164
- Regularizing time series (also Laplacian regularization) page 317 VMLS
- Estimating a periodic time series page 318 VMLS (example of regularization)
- Example of periodic time series to work on page 15.10 VMLS
- de meaned return time series page 378 VMLS
- feature matrix page 257 VMLS

7.2 Least Squares Data Fitting

- "Least squares is widely used to construct a mathematical model from some data, experiments, or observations."
- Note how least squares is approached before fitting to data.
- How to relate to a statistical approach
- Is least squares part of the approximation and fitting chapter?

7.3 Recurrent Neural Networks

Source: Intro Statistical Learning with Python

- Used when data arise as sequences (so we are considering time series data)
- RNNs build models that take into account this sequential nature of the data and build a memory of the past
- The feature for each observation is a sequence of vectors

$$x_t \in \mathbf{R}^n, \quad t = 1, \ldots, L$$

Chapter 8

Statistical Estimation

Unlike in the probability chapter, we

8.1 Parametric Distribution Estimation

My notation

- likelihood function will use the same notation as the probability density function: $p_\theta(x)$.

8.1.1 Maximum Likelihood

Poisson Maximum Likelihood

Suppose we are a store owner. It is obviously desireable to have an estimate for the number of customers that might come to our store on any given day. In the Approximation and Fitting chapter, we addressed this type of problem with **forecasting**. More specifically, if we had a dataset containing the number of customers who came into our store each day, we could fit a time series model (perhaps an AR model or a model with trend and seasonal components) to attempt to predict the customer demand over some time horizon (the next week, perhaps).

In this chapter, we take an alternative approach. Specifically, $y \approx f(x)$. (Although it is worth mentioning that more often than not, forecasting problems)

While **predicting or forecasting demand** is a difficult problem with complications such as seasonality, perhaps we do not yet have any demand data and we just want some baseline model which can help inform our inventory ordering. Furthermore, for the next N days we count the number of customers who visit our store *each day*. We can pose our estimation problem as follows:

- Let $x \in \mathbf{R}$ be a random variable representing the number of customers who visit the store on any given day.
- Since the Poisson distribution is often used to count the number of random arrivals to a system within a given amount of time, we assume that $x \sim \text{Poisson}(\lambda)$.

- Having collected N instances of x

$$p_\lambda(x) = \prod_{t=1}^{T} \frac{e^{-\lambda}\lambda_t^x}{x_t!}$$

The corresponding log-likelihood function

$$\begin{aligned} l(\lambda) &= \left(\sum_{t=1}^{T} -\lambda + x_t \log\lambda - \log(x_t!)\right) \\ &= -T\lambda + \log\lambda \left(\sum_{t=1}^{T} x_t\right) - \left(\sum_{t=1}^{T} \log(x_t!)\right) \end{aligned}$$

$$\text{maximize } -T\lambda + \log\lambda \sum_{t=1}^{T} x_t - \sum_{t=1}^{T} \log(x_t!)$$

Equivalent to (and have written λ under maximize to emphasize). Also note that unlike in other contexts, we have not specified that $\lambda > 0$. I want to stress the reason: we have adopted the notation of CVX and use problem domain. $\lambda > 0$ is not really a constraint on our maximization problem, if you were to give this problem a non-positive lambda, we couldn't even evaluate the problem.

$$\underset{\lambda}{\text{maximize}} \ -T\lambda + \left(\sum_{t=1}^{T} x_t\right) \log\lambda.$$

$$\hat{\lambda} = \text{argmax}_\lambda \ l(\lambda) = \frac{1}{T}\sum_{t=1}^{T} x_t$$

which of course is just the sample mean of the T observations in the sample.

Chapter 9

Geometric and Classification Problems

[BV24]**Exercise 8.8**. *Bounding object position from multiple camera views.* $x \in \mathbf{R}^3$, and viewed by a set of m cameras. Our goal is to find a box in $\mathbf{R}^3$,

$$\mathcal{B} = \left\{ z \in \mathbf{R}^3 \mid l \preceq z \preceq u \right\},$$

for which we can guarantee $x \in \mathcal{B}$. We want the smallest possible such bounding box. (Although it doesn't matter, we can use volume to judge 'smallest' among boxes.)
Now we describe the cameras. The object at location $x \in \mathbf{R}^3$ creates an image on the image plane of camera i at location

$$v_i = \frac{1}{c_i^T x + d_i} (A_i x + b_i) \in \mathbf{R}^2.$$

The matrices $A_i \in \mathbf{R}^{2\times 3}$, vectors $b_i \in \mathbf{R}^2$ and $c_i \in \mathbf{R}^3$, and real numbers $d_i \in \mathbf{R}$ are known, and depend on the camera positions and orientations. We assume that $c_i^T x + d_i > 0$. The 3×4 matrix

$$P_i = \left[\begin{array}{cc} A_i & b_i \\ c_i^T & d_i \end{array} \right]$$

is called the camera matrix (for camera i). It is often (but not always) the case the that the first 3 columns of P_i (i.e., A_i stacked above c_i^T) form an orthogonal matrix, in which case the camera is called orthographic. We do not have direct access to the image point v_i; we only know the (square) pixel that it lies in. In other words, the camera gives us a measurement $\hat{v}_i$ (the center of the pixel that the image point lies in); we are guaranteed that

$$\|v_i - \hat{v}_i\|_\infty \leq \rho_i/2,$$

where ρ_i is the pixel width (and height) of camera i. (We know nothing else about v_i; it could be any point in this pixel.) Given the data $A_i, b_i, c_i, d_i, \hat{v}_i, \rho_i$, we are to find the smallest box $\mathcal{B}$ (i.e., find the vectors l and u) that is guaranteed to contain x. In other words, find the smallest box in $\mathbf{R}^3$ that contains all points consistent with the observations from the camera.

(a) Explain how to solve this using convex or quasiconvex optimization. You must explain any transformations you use, any new variables you introduce, etc. If the convexity or quasiconvexity of any function in your formulation isn't obvious, be sure justify it.

(b) Solve the specific problem instance given in the file `camera_data.m`. Be sure that your final numerical answer (i.e., l and u) stands out.

Response. Let's summarize the prompt and understand the problem .

x is some object in the physical world (modeled as 3-dimensional Euclidean space). We have m cameras which capture images of x. However, because

Let's first formulate the constraints ensuring that our proposed box that x lies in is consistent with the camera measurements; *i.e.*, there isn't a point in the bounding box which contra

$$\|v_i - \hat{v}_i\|_\infty \leq \rho_i/2,, \quad i = 1, \dots, m$$

is expanded as

$$\left\| \frac{1}{c_i^T x + d_i} (A_i x + b_i) - \hat{v}_i \right\|_\infty \leq \rho_i/2, \quad i = 1, \dots, m.$$

The ℓ_∞-norm is requiring that the absolute value of both terms in the $\mathbf{R}^2$ vector

$$\frac{1}{c_i^T x + d_i} (A_i x + b_i) - \hat{v}_i$$

be less than or qual to $\rho_i/2$. Furthermore,

$$-(\rho_i/2)\mathbf{1} \preceq \frac{1}{c_i^T x + d_i} (A_i x + b_i) - \hat{v}_i \preceq (\rho_i/2)\mathbf{1}, \quad i = 1, \dots, m$$

$$(\hat{v}_i - (\rho_i/2)\mathbf{1})(c_i^T x + d_i) \preceq A_i x + b_i \preceq (\hat{v}_i + (\rho_i/2))(c_i^T x + d_i), \quad i = 1, \dots, m$$

Part III

Applications Part II

Chapter 10

Control

10.1 Overview

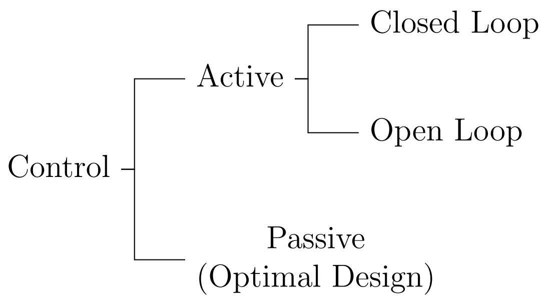

Figure 10.1: Control Strategies

$$\begin{tikzcd} \boxed{\text{Controller}} \arrow[r, "u_t"] & \boxed{\text{System}} \arrow[r, "x_{t+1}"] & {} \end{tikzcd}$$

Figure 10.2: Open Loop Control Strategy

Source: Control bootcamp (Brunton)

- (Dynamical Systems) Systems of ODEs describing state of system has been a successful modeling framework.
- Often want to actively make changes to the system.
- In control theory, you have a dynamical system of interest, you write down the system of equations which describes the behavior of the system, and then you create a control theory to create a more “desireable” system behavior.
- Passive control (Boyd would call optimal design): design an upfront solution (ex: minimizing drag on a truck)

- Active control: pump energy into the system to actively manipulate its behavior.

Active Control

- Open Loop

10.2 Optimal Design

Note to Readers: The following exercise uses Geometric Programming (GP). Please note that the language of GPs has their own definition of *monomials*, which does not agree with the algebraic definition of a monomial. There is also an object known as a *posynomial.*

[BV24] **Exercise 18.14**. *Design of an unmanned aerial vehicle.* You are tasked with developing the high-level design for an electric unmanned aerial vehicle (UAV). The goal is to design the least expensive UAV that is able to complete K missions, labeled $k = 1, \ldots, K$. Mission k involves transporting a payload of weight $W_k^{\text{pay}} > 0$ (in kilograms) over a distance $D_k > 0$ (in meters), at a speed $V_k > 0$ (in meters per second). These mission quantities are given. The high-level design consists of choosing the engine weight W^{eng} (in kilograms), the battery weight W^{bat} (in kilograms), and the wing area S (in m^2), within the given limits

$$W_{\min}^{\text{eng}} \leq W^{\text{eng}} \leq W_{\max}^{\text{eng}}, \quad W_{\min}^{\text{bat}} \leq W^{\text{bat}} \leq W_{\max}^{\text{bat}}, \quad S_{\min} \leq S \leq S_{\max}.$$

(The lower limits are all positive.) We refer to the variables W^{eng} , W^{bat} , and S as the design variables. In addition to choosing the design variables, you must choose the power $P_k > 0$ (in watts) that flows from the battery to the engine, and the angle of attack $\alpha_k > 0$ (in degrees) of the UAV during mission k, for $k = 1, \ldots, K$. These must satisfy

$$0 \leq P_k \leq P_{\max}, \quad 0 \leq \alpha_k \leq \alpha_{\max},$$

where $\alpha_{\max}$ is given, and $P_{\max}$ depends on the engine weight as described below. We refer to these $2K$ variables as the mission variables. The engine weight, battery weight, and wing area are the same for all k missions; the power and angle of attack can change with the mission. The weight of the wing is W^{wing} (in kilograms) is given by $W^{\text{wing}} = C_W S^{1.2}$, where $C_W > 0$ is given. The total weight of the UAV during mission k, denoted W_k, is the sum of the battery weight, engine weight, wing weight, the payload weight, and a baseline weight W^{base} , which is given. The total weight depends on the mission, via the payload weight, and so is subscripted by k. The lift and drag forces acting on the UAV in mission k are

$$F_k^{\text{lift}} = \frac{1}{2}\rho V_k^2 C_L(\alpha_k) S, \quad F_k^{\text{drag}} = \frac{1}{2}\rho V_k^2 C_D(\alpha_k) S$$

(in newtons), where C_L and C_D are the lift and drag coefficients as functions of the angle of attack α_k, and $\rho > 0$ is the (known) air density (in kilograms per cubic meter). We will use the simple functions

$$C_L(\alpha) = c_L \alpha, \quad C_D(\alpha) = c_{D1} + c_{D0}\alpha^2,$$

where $c_L > 0, c_{D0} > 0$, and $c_{D1} > 0$ are given constants. To maintain steady level flight, the lift must equal the weight, and the drag must equal the thrust from the propeller, denoted T_k (in newtons), i.e.,

$$F_k^{\text{lift}} = W_k, \quad F_k^{\text{drag}} = T_k.$$

The thrust force, power P_k (in watts), and the UAV speed are related via $P_k = T_k V_k$. The engine maximum power is related to its weight by $W^{\text{eng}} = C_P P_{\text{max}}^{0.803}$ where $C_P > 0$ is given. The battery capacity E (in joules) is equal to $C_E W^{\text{bat}}$, where $C_E > 0$ is given. The total energy expended over mission k, with speed V_k, power output P_k, and distance D_k is $P_k D_k / V_k$. This must not exceed the battery capacity E. The overal cost of the UAV is the sum of a design cost and a mission cost. The design cost C_{des}, which is an approximation of the cost of building the UAV, is given by

$$C_{\text{des}} = 100 W^{\text{eng}} + 45 W^{\text{bat}} + 2 W^{\text{wing}}.$$

The mission cost C_{mis} is given by

$$C_{\text{mis}} = \sum_{k=1}^{K} (T_k + 10\alpha_k),$$

which captures our desire that the thrust and angle of attack be small. In summary, $W_{\text{min}}^{\text{eng}}, W_{\text{max}}^{\text{eng}}, W_{\text{min}}^{\text{bat}}, W_{\text{max}}^{\text{bat}}, S_{\text{min}}, S_{\text{max}}, \alpha_{\text{max}}, W_{\text{base}}, C_W, c_L, c_{D0}, c_{D1}, C_P, C_E,$ and ρ are given. Additionally, D_k, V_k, and W_k^{pay} are given for $k = 1, \ldots, K$.

(a) The problem as stated is almost a geometric problem (GP). By relaxing two constraints it becomes a GP, and therefore readily solved. Identify these constraints and give the relaxed versions. Briefly explain why the relaxed constraints will be tight at the solution, which means by solving the GP, you've actually solved the original problem. You do not need to reduce the relaxed problem to a standard form GP, or the equivalent convex problem; it's enough to express it in DGP compatible form.

(b) Solve the relaxed problem you formulate in part (a) with data given in the provided python file. Give the optimal costs C_{des}^* and C_{mis}^*, and the values of all design and mission variables. Check that at your solution the relaxed constraints are tight.

Response. Recall that an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \le 1, \quad i = 1, \ldots, m \\ & h_i(x) = 1, \quad i = 1, \ldots, p, \end{array}$$

where $f_0, \ldots f_m$ are posynomials and $h_1, \ldots h_p$ are monomials is a *geometric program* (GP) in *standard form.* (Note that $\mathcal{D} = \mathbf{R}_{++}^n$; the constraint $x \succ 0$ is implicit.)

Most importantly, note that the equality constraint functions must be monomials. This requirement helps direct our search for invalid constraints, *i.e.*, we should look for constraints, which as formulated "verbally," would require posynomial equalities. The first such constraint(s) is

$$F_k^{\text{lift}} = W_k, \quad k = 1, \ldots, K,$$

since

$$\begin{aligned} W_k &= W^{\text{bat}} + W^{\text{eng}} + W^{\text{wing}} + W_k^{\text{pay}} + W^{\text{base}} \\ &= W^{\text{bat}} + W^{\text{eng}} + C_W S^{1.2} + W_k^{\text{pay}} + W^{\text{base}}, \end{aligned}$$

is a *posynomial* and F_k^{lift} is a *monomial*. Because posynomials are *closed under division*, the proposed constraint

$$F_k^{\text{lift}} = W_k \iff \left(F_k^{\text{lift}}\right)^{-1} W_k = 1, \quad k = 1, \ldots, K,$$

is a posynomial equality constraint, which to emphasize again, is invalid for GP formulation.

For the same reason, the constraint(s)

$$F_k^{\text{drag}} = T_k, \quad k = 1, \ldots, K,$$

is also invalid. However, in this case, it is the force term, F_k^{drag}, which is the posynomial and $T_k = (1/V_k)P_k$ which is the monomial. To formulate a proper GP, we therefore make the following two (or more accurately, $2K$) relaxations

$$W_k = F_k^{\text{lift}}, \quad \text{and} \quad F_k^{\text{drag}} = T_k, \quad k = 1, \ldots, K$$

become

$$W_k \leq F_k^{\text{lift}}, \quad \text{and} \quad F_k^{\text{drag}} \leq T_k, \quad k = 1, \ldots, K,$$

or equivalently (and in GP standard form)

$$\left(F_k^{\text{lift}}\right)^{-1} W_k \leq 1, \quad \text{and} \quad T_k^{-1} F_k^{\text{drag}} \leq 1, \quad k = 1, \ldots, K,$$

in the relaxed formulation.

It is now highly tempting to start writing down an optimization problem by just reading off the problem specifications. *This is discouraged.* Even though we have found and proposed relaxations for the two originally invalid GP constraints, it is still *very easy* to formulate a *non-DGP compatible problem.* As an example, a common (and understandable) mistake would be to include the constraints

$$0 \leq P_k, \quad k = 1, \ldots, K,$$

in the formulation. However, **including them** would make the formulation **non-DGP compatible**. All constraints, equality or inequality, must have a 1 on the right-hand side of the constraint when reduced to standard form. The constraints $0 \leq P_k$, $k = 1, \ldots, K$ cannot be reduced to this form. However, these are valid GP constraints! They are just included *implicitly* since the domain of the problem is $\mathbf{R}_{++}^n$. Including them *explicitly* leads to a non-DGP compatible formulation.

Additionally, unlike in LPs, where additional "linking" constraints are harmless, adding such syntactic constraints here can, again, lead to a non-DGP compatible problem. As an example, consider the specification that the lift force acting on the UAV in mission k is

$$F_k^{\text{drag}} = \frac{1}{2}\rho V_k^2 C_D(\alpha_k) S,$$

which as previously discussed is a *posynomial.* In an LP formulation (at least, as commonly taught in an operations research setting), it is encouraged (and would be totally valid) to declare F_k^{drag} as an optimization variable, include the above equality in the LP **as a constraint**, and then use F_k^{drag} throughout the remainder of the problem in constraints involving drag force. However, this "linking" constraint is **non-DGP compatible** because it would be reduced to a posynomial equality constraint, which as discussed at greath length above, is **not a valid GP constraint**.

Furthermore, we must distinguish between actual *optimization variables* and what I'll call *derived optimization variables.* The former are what they are always defined as: the variables that are to be chosen to minimize the objective function. We will define the latter as a mathematical expression whose value is decided by the optimization variables. To use a Computer Science dialect, the derived optimization variables are functions of references to the optimization variables (where the function can take multiple optimization variables and even other derived optimization variables). Consequently, we can use either the derived variables or the actual variables when formulating our problem. As an example, the constraint

$$\frac{1}{2}\rho V_k^2 C_D(\alpha_k) S \leq (1/V_k) P_k, \quad k = 1, \ldots, K$$

is equivalent to

$$F_k^{\text{drag}} \leq T_k, \quad k = 1, \ldots, K.$$

However, $F_k^{\text{drag}} = \frac{1}{2}\rho V_k^2 C_D(\alpha_k) S$ and $T_k = (1/V_k) P_k$, $k = 1, \ldots, K$, should **not** be included in the problem constraints and F_k^{drag}, T_k, $k = 1 \ldots, K$ **are not** optimization variables. They both are merely expressions *encoding* optimization variables. As a final attempt to stress the difference, when building this problem with a declarative language such as CVXPY in Python, the problem should be created as follows (and see Algorithm 1 for the actual code)

1. Declare optimization variables. Ex. `opt_var = cvxpy.Variable(num_entries)` (notice the explicit use of the `Variable` keyword).

2. Create derived optimization variables. Ex. `der_var = 2.5*opt_var` (this becomes a `cvxpy.Expression`).

3. Create the constraints and objective using the optimization variables, the derived optimization variables, or some combination of the two. Ex. `der_var <= 5`, which would be equivalent to adding the constraint `opt_var <= 2`.

4. Solve the problem.

This is a nuanced point, and perhaps is obvious to readers who do not have training in OR formulating LPs, but it is critically important.

We now put everything together; the following is the Relaxed UAV Design Problem.

Optimization Variables

- Design Variables: W^{eng}, W^{bat}, S.
- Mission Variables: P_k, α_k, $\quad k = 1, \ldots, K$.

Derived Variables/CVXPY expressions

- $W^{\text{wing}} = C_W S^{1.2}$,
- $W_k = W^{\text{bat}} + W^{\text{eng}} + W^{\text{wing}} + W_k^{\text{pay}} + W^{\text{base}}$,
- $F_k^{\text{lift}} = \frac{1}{2}\rho V_k^2 C_L(\alpha_k) S, \quad k = 1, \ldots, K$,
- $F_k^{\text{drag}} = \frac{1}{2}\rho V_k^2 C_D(\alpha_k) S, \quad k = 1, \ldots, K$,
- $P_{\max} = \left(C_P^{-1} W^{\text{eng}}\right)^{1/0.803}$,
- $T_k = (1/V_k) P_k, \quad k = 1, \ldots, K$,
- $E = C_E W^{\text{bat}}$,
- $C_{\text{des}} = 100 W^{\text{eng}} + 45 W^{\text{bat}} + 2 W^{\text{wing}}$,
- $C_{\text{des}} = \sum_{k=1}^{K} (T_k + 10\alpha_k)$.

Relaxed Formulation

$$\begin{array}{ll}
\text{minimize} & C_{\text{des}} + C_{\text{mis}} \\
\text{subject to} & W_{\min}^{\text{eng}} \leq W^{\text{eng}} \leq W_{\max}^{\text{eng}}, \quad W_{\min}^{\text{bat}} \leq W^{\text{bat}} \leq W_{\max}^{\text{bat}}, \quad S_{\min} \leq S \leq S_{\max} \\
& P_k \leq P_{\max}, \quad \alpha_k \leq \alpha_{\max}, \quad k = 1, \ldots, K \\
& F_k^{\text{lift}} \geq W_k, \quad k = 1, \ldots K \\
& F_k^{\text{drag}} \leq T_k, \quad k = 1, \ldots K \\
& (1/V_k) P_k D_k \leq E, \quad k = 1, \ldots K.
\end{array}$$

Problem Solution and Analysis.

Firstly, we address why the relaxed constraints, $F_k^{\text{lift}} \geq W_k$ and $F_k^{\text{drag}} \leq T_k$, $k = 1, \ldots, K$, will be tight at the solution.

The tightness in the constraint $F_k^{\text{drag}} \leq T_k$ at the solution is easy to see. We proceed with a semi-formal argument. Suppose that the drag inequality for mission k holds strictly. That is, $F_k^{\text{drag}} < T_k$. If this is the case, then we can reduce the thrust for mission k by decreasing the power, P_k, that flows from the battery to the engine during that mission as T_k strictly increases and strictly decreases as P_k increases or decreases. We are able to decrease the power because the only lower bound on P_k is the implicit lower bound $P_k \geq 0$, and if it is the case that $P_k \to 0$, then F_k^{drag} must also go to zero since it too is positive. As both terms

go to zero, we end up violating the original assumption that F_k^{drag} is strictly less than the thrust. Furthermore, being able to reduce T_k contradicts the assumption that we are at the solution since shrinking T_k lowers the mission cost, and thus the overall cost.
A nearly identical argument can be made for the tightness in the constraint F_k^{lift} with respect to the attack angle optimization variable α_k.

(b) The Python code in Algorithm 1 contains the core Python code required to compute the optimal UAV design (it doesn't include constant instantiations and helper functions). The corresponding optimal values are

- $C^*_{\text{des}} = 4449.324$,
- $C^*_{\text{mis}} = 1556.587$,
- $W^{\text{eng}} = 9.951\,\text{kg}$,
- $W^{\text{bat}} = 74.06\,\text{kg}$,
- $S = 7.999\,\text{m}^2$,
- the power values are

$$P_1 = 40.159\,\text{kW}, \quad P_2 = 12.142\,\text{kW}, \quad P_3 = 20.018\,\text{kW},$$
$$P_4 = 24.998\,\text{kW}, \quad P_5 = 8.080\,\text{kW},$$

- and the attack angles are

$$\alpha_1 = 0.1377^\circ, \quad \alpha_2 = 0.3141^\circ, \quad \alpha_3 = 0.2249^\circ, \quad \alpha_4 = 0.2114^\circ, \quad \alpha_5 = 0.3785^\circ.$$

Finally, in table 10.1 we see that the relaxed constraints are indeed tight.

k	T_k	F_k^{drag}	F_k^{lift}	W_k
1	489.739 N	489.739 N	269.648 N	269.646 N
2	220.763 N	220.763 N	276.647 N	276.646 N
3	307.969 N	307.969 N	276.648 N	276.646 N
4	357.119 N	357.119 N	301.648 N	301.646 N
5	168.332 N	168.332 N	253.946 N	253.946 N

Table 10.1: Tightness in Relaxed Constraints.

Algorithm 1: UAV Design Python Code

```python
        ### optimization variables ###
        W_eng = cp.Variable(1, pos=True)
        W_bat = cp.Variable(1, pos=True)
        S = cp.Variable(1, pos=True)
        P = cp.Variable(K, pos=True)
        alpha = cp.Variable(K, pos=True)

        ### derived expressions ###
        W_wing = CW*S**(1.2)
        W_k = [W_bat + W_eng + W_wing + W_base + W_pay[k] for k in range(K)]
        P_max = ((1/CP)**(1/0.803)) * (W_eng**(1/0.803))
        F_k_lift = [0.5*rho* V[k]**2 * C_L(alpha[k]) * S for k in range(K)]
        F_k_drag = [0.5*rho * V[k]**2 * C_D(alpha[k])*S for k in range(K)]
        T_k = [P[k] * (1/V[k]) for k in range(K)]
        E = CE*W_bat
        C_des = 100*W_eng + 45*W_bat + 2 * W_wing
        C_mis = cp.sum([T_k[k] + 10*alpha[k] for k in range(K)])
        ### ###

        ### constraints and objective ###
        constraints = [W_eng_min <= W_eng, W_eng <= W_eng_max,
                    W_bat_min <= W_bat, W_bat <= W_bat_max,
                    S_min <= S, S <= S_max]
        constraints += [constr for constr_tuple in [(P[k] <= P_max,
                        alpha[k] <= alpha_max) for k in range(K)]
                        for constr in constr_tuple]
        constraints += [F_k_lift[k] >= W_k[k] for k in range(K)]
        constraints += [F_k_drag[k] <= T_k[k] for k in range(K)]
        constraints += [P[k] * D[k] * (1/V[k]) <= E for k in range(K)]

        obj = cp.Minimize(C_des + C_mis)
        prob = cp.Problem(obj, constraints)
        # print(prob.is_dgp())
        prob.solve(gp=True)

```

10.3 Basic Examples

[BV04] **Exercise 4.16**. *Minimum fuel optimal control.* Consider the LTI dynamical system with state $x_t \in \mathbf{R}^n$, $t = 0, \dots, N$, and actuator or input signal $u_t \in \mathbf{R}$, for $t = 0, \dots, N-1$. The dynamics of the system are governed by the linear recurrence

$$x_{t+1} = Ax_t + bu_t, \quad t = 0, \dots N-1,$$

where $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^n$ are given. Assume the initial state is $x_0 = 0$. The *minimum fuel optimal control problem* is to choose the inputs $u_0, \ldots u_{N-1}$ so as to minimize the total fuel consumed, which is given by

$$F = \sum_{t=0}^{N-1} f(u_t),$$

subject to the constraint that $x_N = x_{\text{des}}$, where N is the (given) time horizon and $x_{\text{des}} \in \mathbf{R}^n$ is the (given) desired final or target state. The function $f : \mathbf{R} \to \mathbf{R}$ is the *fuel use map* for the actuator, and gives the amount of fuel used as a function of the actuator signal amplitude. In this problem we use

$$f(a) = \begin{cases} |a| & |a| \leq 1 \\ 2\,|a| - 1 & |a| > 1. \end{cases}$$

Formulate the minimum fuel control problem as an LP.

Response. Firstly, a few notes about the problem itself:

- [Rea20] What is an actuator (broadly)?
  - A device that makes something move or operate. As a trivial example, an actuator moves the sliding doors at a grocery store.
  - Two types: straight line movement (linear) and circular movement (rotary).
  - An actuator converts a source of energy into a physical, mechanical motion. A silly example of this: a handwheel can be used to feed energy into a rotary actuator. In industrial applications, there are three typical sources of energy: *Electric* uses electricty (duh), *hydraulic* use a variety of liquids, *pneumatic* are operated by compressed air.
  - Common industrial actuators are electric motors, hydraulic motors, and pneumatic control valves.
  - Example use of a pneumatic actuator: "PLC analog output card* produces a 4-20 mA current to move the valve from fully open to fully closed. The 4-20 mA current will be converted to pneumatic pressure, which becomes the source of energy to operate the actuator."
  - *A Programmable Logic Controller (PLC) analog output card is a component used to convert digital signals from the PLC's processor into analog signals (digital-to-analog conversion, DAC that can be used to control external devices.
- Actuator in this problem.
  - u_t is a scalar valued signal that directly affects the state of our system of interest (since it's in the linear recurrence equation).
  - Physically, this signal is being used to direct the actuator, which in turn is turning energy into mechanical motion.

- Therefore, the fuel use map for the actuator is a mathematical relation between the amplitude of the signal directing the actuator and the fuel being used. In other words, we can imagine a larger actuator signal corresponds to greater actuator motion, and in this instance, the source of energy needed to produce this motion is "fuel."

We want to write this minimum fuel optimal control problem as a LP. However, a LP formulation is rather restrictive, so let's begin by formulating this problem as a convex optimization problem. It is tempting to naively claim that the problem

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{N-1} f(u_t) \\ \text{subject to} & x_{t+1} = Ax_t + bu_t, \qquad t = 0, \dots, N-1 \\ & x_0 = 0, \quad x_N = x_{\text{des}}. \end{array}$$

is convex. However, we must remember that f is a piecewise function that we are unfamiliar with (*e.g.* it isn't a piecwise linear function). Furthermore, this proprosed formulation is not a conex optimization problem. Of course, observing that the components of f are indeed convex, it seems highly plausible there's a way that f can be reformulated such that the optimization problem is convex. (Note that this is similar to having to formulate the Huber penalty approximation problem as a convex problem, [BV04] **Exercise 6.2**.) Consider the graph of the fuel use map in figure 10.3. As stated in the legend of this figure, and clearly seen when observing the graph, the piecewise function f is equivalent to $\max\{|a|, 2|a| - 1\}$. (Of course one could also provide an algebraic argument if so desired.) Furthermore, the optimal fuel control problem can be formulated as

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{N-1} \max\{|u_t|, 2|u_t| - 1\} \\ \text{subject to} & x_{t+1} = Ax_t + bu_t, \qquad t = 0, \dots, N-1 \\ & x_0 = 0, \quad x_N = x_{\text{des}}, \end{array}$$

which is a valid convex optimization problem. If our goal was simply to find the optimal solution to this fuel problem and we didn't care about formulating the problem as a LP, **we could end our reformulating here.** In fact, the optimal solution $u^* \in \mathbf{R}^{Nm}$, where $Nm = N1 = N$, plotted in figure 10.4 was obtained by solving this convex problem. Nonetheless, we trudge forward with our LP formulation.
For the time being, to be more concise, let's drop the linear dynamical system constraints

$$x_0 = 0, \; x_N = x_{\text{des}}, \; \text{ and } \; x_{t+1} = Ax_t + bu_t, \quad t = 0, \dots N-1$$

and just consider the unconstrained problem

$$\text{minimize} \sum_{t=0}^{N-1} \max\{|u_t|, 2|u_t| - 1\}.$$

It is highly tempting to attempt to map the objective to a ℓ_1- or ℓ_∞-norm approximation problem and use the reformulation techniques detailed in 4.3.1, however, the objective function is neither a sum of (pure) absolute values (like $\|Ax - b\|_1$) nor the maximum of absolute

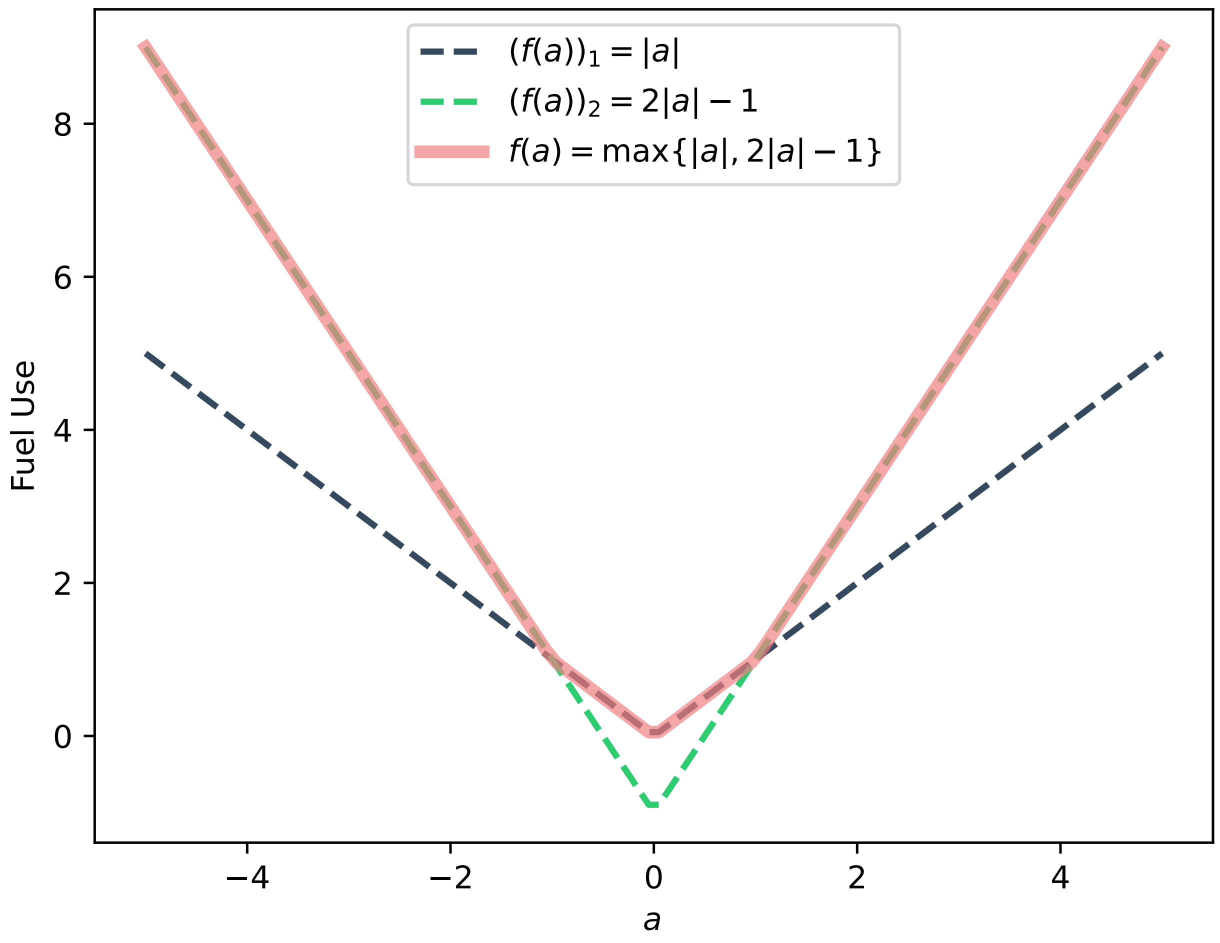

Figure 10.3: Actuator Fuel Use Map.

values (like $\|Ax - b\|_\infty$). It is instead a sum of maximums of linear functions of absolute values. Furthermore, while there is perhaps a way to map f_0 directly into some composition of the two norms, instead of trying to use these "atoms" directly, we can propose a reformulation for our convex problem using the same derivation *techniques* discussed in 4.3.1.

Our first reformulation uses that an objective function defined as the sum of absolute value/maximum expressions can be rewritten as a sum of auxilary variables, $s \in \mathbf{R}^N$ here, with each summand being less than or equal to an element in the auxilary variable vector. To simplify the problem further we can also remove the max operator from each summand using that if the maximum element in the set being operated on by max is less than or equal to s_t, then so must every other element. These two reformulation techniques yield the problem

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s \\ \text{subject to} & |u_t| \leq s_t, \qquad t = 1, \dots N \\ & 2\,|u_t| - 1 \leq s_t, \quad t = 1, \dots N. \end{array}$$

To fully linearize the problem, note that if the constraints

$$2\,|u_t| - 1 \leq s_t, \quad t = 1, \dots N$$

were not present then

$$|u_t| \leq s_t, \quad t = 1, \dots N$$

could just be reformulated as $-s \preceq u \preceq s$. Furthermore, thinking again about how in the reformulations derived in 4.3.1 we introduced new auxilary variables to remove nonlinearities from expressions, consider $y \in \mathbf{R}^N$ and the problem

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s \\ \text{subject to} & y \preceq s \\ & 2y - \mathbf{1} \preceq s \\ & -y \preceq u \preceq y, \end{array}$$

which uses this new variable to "pull out" the absolute value from the two other sets of constraints and then is restricted in the same way we would've reformulated $|u_t| \leq s_t$. Now, returning our LDS constraints to the formulation we have the problem

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s \\ \text{subject to} & x_{t+1} = Ax_t + bu_t, \qquad t = 0, \dots, N-1 \\ & x_0 = 0, \quad x_N = x_{\text{des}} \\ & y \preceq s \\ & 2y - \mathbf{1} \preceq s \\ & -y \preceq u \preceq y; \end{array}$$

however, note that while all problem functions are linear (affine), the LDS constraints are still in the form of a linear recurrence. Observing the pattern

$$\begin{aligned} x_1 &= Ax_0 + bu_0 \\ x_2 &= Ax_1 + bu_1 \\ &= A(Ax_0 + bu_0) + bu_1 = A^2 x_0 + Abu_0 + bu_1 \\ x_3 &= A^3 x_0 + A^2 b_0 + Abu_1 + bu_2 \\ &\ \vdots \end{aligned}$$

and plugging in the values for x_0 and x_{des} into the recurrence, we see that the LDS constraints can be replaced with the affine equality constraints given by

$$Hu = x_{\text{des}},$$

where

$$H = \begin{bmatrix} A^{N-1}b & A^{N-2}b & \cdots & Ab & b \end{bmatrix}.$$

Therefore, our original minimum fuel control problem is equivalent to

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s \\ \text{subject to} & Hu = x_{\text{des}} \\ & y \preceq s \\ & 2y - \mathbf{1} \preceq s \\ & -y \preceq u \preceq y, \end{array}$$

which is a LP.

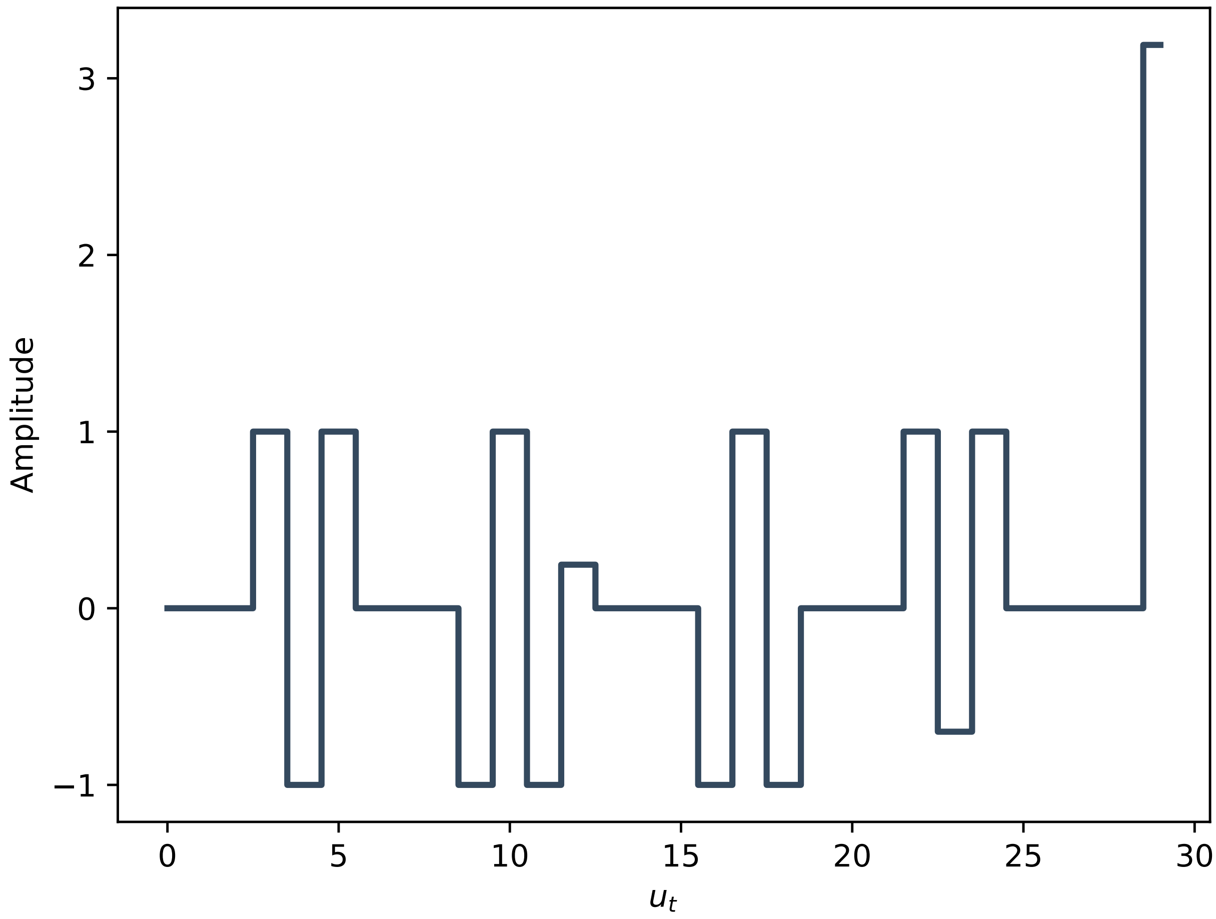

Figure 10.4: Minimum fuel actuator signal.

(Note that in the following, $e_3 = (0, 0, 1)$.)
[BV24]**Exercise 16.2**. *Optimal Spacecraft Landing.* We consider the problem of optimizing the thrust profile for a spacecraft to carry out a landing at a target position. The spacecraft dynamics are

$$m\ddot{p} = f - mge_3,$$

where $m > 0$ is the spacecraft mass, $p(t) \in \mathbf{R}^3$ is the spacecraft position, with 0 the target landing position and $p_3(t)$ representing height, $f(t) \in \mathbf{R}^3$ is the thrust force, and $g > 0$ is the gravitational acceleration. (For simplicity we assume that the spacecraft mass is constant. This is not always a good assumption, since the mass decreases with fuel use. We will also ignore any atmospheric friction.) We must have $p\left(T^{\mathrm{td}}\right) = 0$ and $\dot{p}\left(T^{\mathrm{td}}\right) = 0$, where T^{td} is the touchdown time. The spacecraft must remain in a region given by

$$p_3(t) \geq \alpha \left\|(p_1(t), p_2(t))\right\|_2 ,$$

where $\alpha > 0$ is a given minimum glide slope. The initial position $p(0)$ and velocity $\dot{p}(0)$ are given.
The thrust force $f(t)$ is obtained from a single rocket engine on the spacecraft, with a given maximum thrust; an attitude control system rotates the spacecraft to achieve any desired direction of thrust. The thrust force is therefore characterized by the constraint $\|f(t)\|_2 \leq F^{\max}$. The fuel use rate is proportional to the thrust force magnitude, so the total fuel use is

$$\int_0^{T^{td}} \gamma \left\|f(t)\right\|_2 dt$$

where $\gamma > 0$ is the fuel consumption coefficient. The thrust force is discretized in time, i.e., it is constant over consecutive time periods of length $h > 0$, with $f(t) = f_k$ for $t \in [(k-1)h, kh)$, for $k = 1, \ldots, K$, where $T^{\mathrm{td}} = Kh$. Therefore we have

$$v_{k+1} = v_k + (h/m)f_k - hge_3, \quad p_{k+1} = p_k + (h/2)\left(v_k + v_{k+1}\right),$$

where p_k denotes $p((k-1)h)$, and v_k denotes $\dot{p}((k-1)h)$. We will work with this discrete-time model. For simplicity, we will impose the glide slope constraint only at the times $t = 0, h, 2h, \ldots, Kh$.

(a) *Minimum fuel descent.* Explain how to find the thrust profile $f_1, \ldots, f_K$ that minimizes fuel consumption, given the touchdown time $T^{\mathrm{td}} = Kh$ and discretization time h.

(b) *Minimum time descent.* Explain how to find the thrust profile that minimizes the touchdown time, i.e., K, with h fixed and given. Your method can involve solving several convex optimization problems.

(c) Carry out the methods described in parts (a) and (b) above on the problem instance with data given in `spacecraft_landing_data.py`. Report the optimal total fuel consumption for part (a), and the minimum touchdown time for part (b). The data files also contain plotting code (commented out) to help you visualize your solution. Use the code to plot the spacecraft trajectory and thrust profiles you obtained for parts (a) and (b).

Remarks. If you'd like to see the ideas of this problem in action, watch these videos:

- Grasshopper Diver, Single Cam
- Grasshopper 24-story Hover
- Falcon 9 Landing at Zone 1

- Falcon 9 First Stage Landing

Response.
(a). Formulating this problem is very straightforward as all control specifications are already DCP compatible, *i.e.*, the specified control problem is convex. After discretizing the integral expression for total fuel use as

$$J_{\text{input}} = \sum_{k=0}^{K-1} \gamma \|f_k\|_2 ,$$

the minimum fuel descent problem can immediately be formulated as

$$\begin{array}{lll}
\text{minimize} & J_{\text{input}} & \\
\text{subject to} & v_{k+1} = v_k(h/m)f_k - hge_3, & k = 0, \dots, K-1 \\
& p_{k+1} = p_k + (h/2)(v_k + v_{k+1}), & k = 0, \dots, K-1 \\
& \alpha \left\| \begin{bmatrix} (p_k)_1 \\ (p_k)_2 \end{bmatrix} \right\|_2 \leq (p_k)_3, & k = 1, \dots, K \\
& \|f_k\|_2 \leq F^{\max}, & k = 0, \dots, K-1 \\
& v_0 = \dot{p}(0), \quad p_0 = p(0) & \\
& v_K = 0, \quad p_K = 0. &
\end{array}$$

(Note that we've adopted a Pythonic zero-based indexing for the formulation, *i.e.*, the first force we apply is at $t = 0$, right when we start controlling the rocket).

(b). Keeping the parameter h fixed, we reduce the time it takes to descend by decreasing K, the number of steps it takes to move from $(p(0), \dot{p}(0)) \to (0, 0)$. The simplest way to find K is with a linear search where we start by solving the feasibility problem,

$$\begin{array}{lll}
\text{minimize} & 0 & \\
\text{subject to} & v_{k+1} = v_k(h/m)f_k - hge_3, & k = 0, \dots, K-1 \\
& p_{k+1} = p_k + (h/2)(v_k + v_{k+1}), & k = 0, \dots, K-1 \\
& \alpha \left\| \begin{bmatrix} (p_k)_1 \\ (p_k)_2 \end{bmatrix} \right\|_2 \leq (p_k)_3, & k = 1, \dots, K \\
& \|f_k\|_2 \leq F^{\max}, & k = 0, \dots, K-1 \\
& v_0 = \dot{p}(0), \quad p_0 = p(0) & \\
& v_K = 0, \quad p_K = 0, &
\end{array}$$

with a K which yields a feasible solution, then solving the same feasibility problem again but with $K^{\text{curr}} = K - 1$. We continue to decrement the planning horizon, K^{curr} by one until the problem becomes infeasible. At this point, we know that $K^{\text{curr}} + 1$ and the associated solution vector $f^* = (f_1, \dots, f_{K^{curr}+1})$ is the minimum number of time steps and associated thrust profile which minimizes the touchdown time.

(c). For part (a), the total fuel consumption is about 193.0. For part (b), we find that $K = 25$ minimizes the touchdown time. The minimum fuel descent trajectory and associated thrust profiles are shown in figure 10.5 and the minimum touchdown time trajectory and associated thrust profiles are shown in figure 10.6.

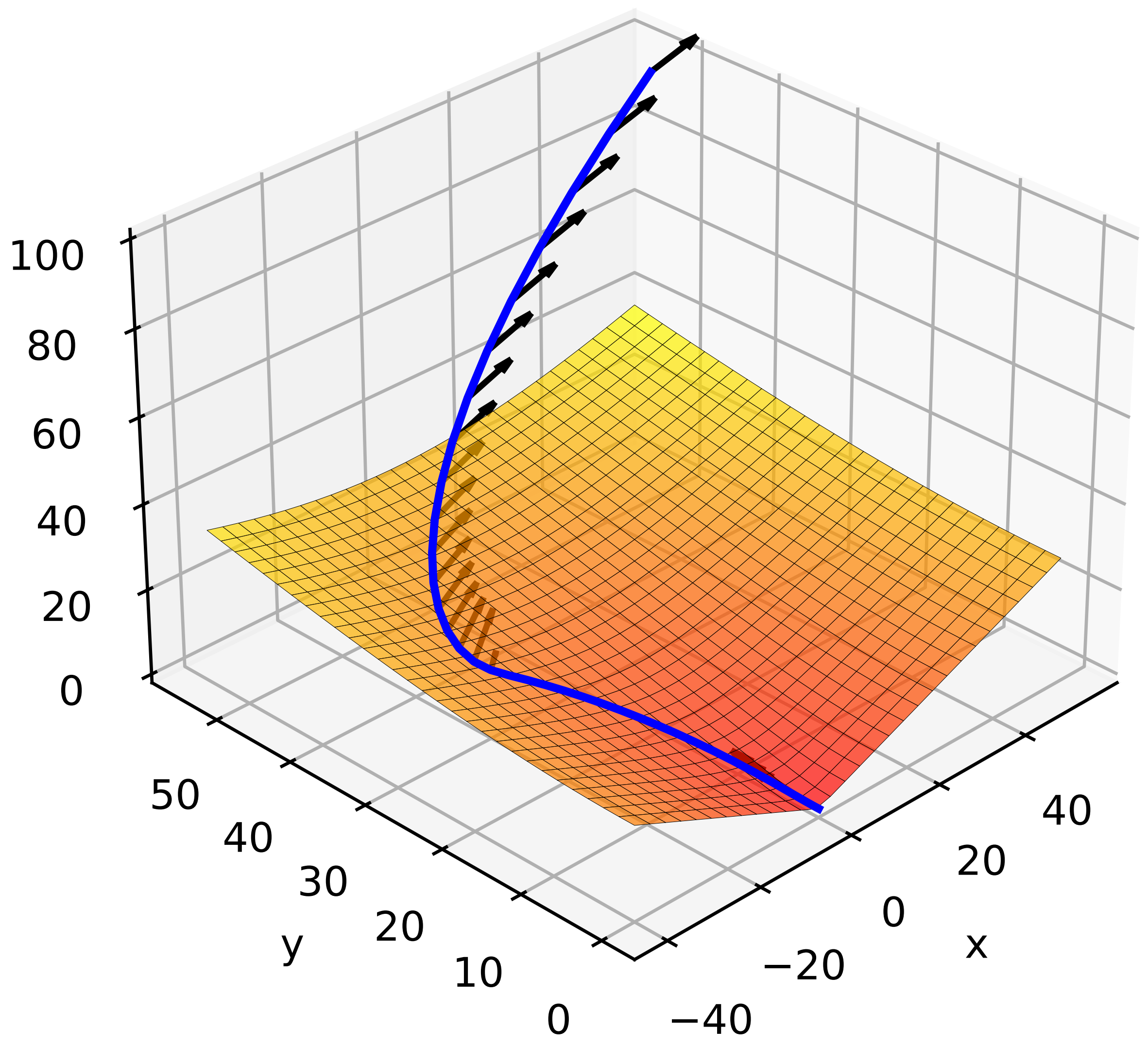

Figure 10.5: Minimum Fuel Descent Trajectory and Thrust Profiles.

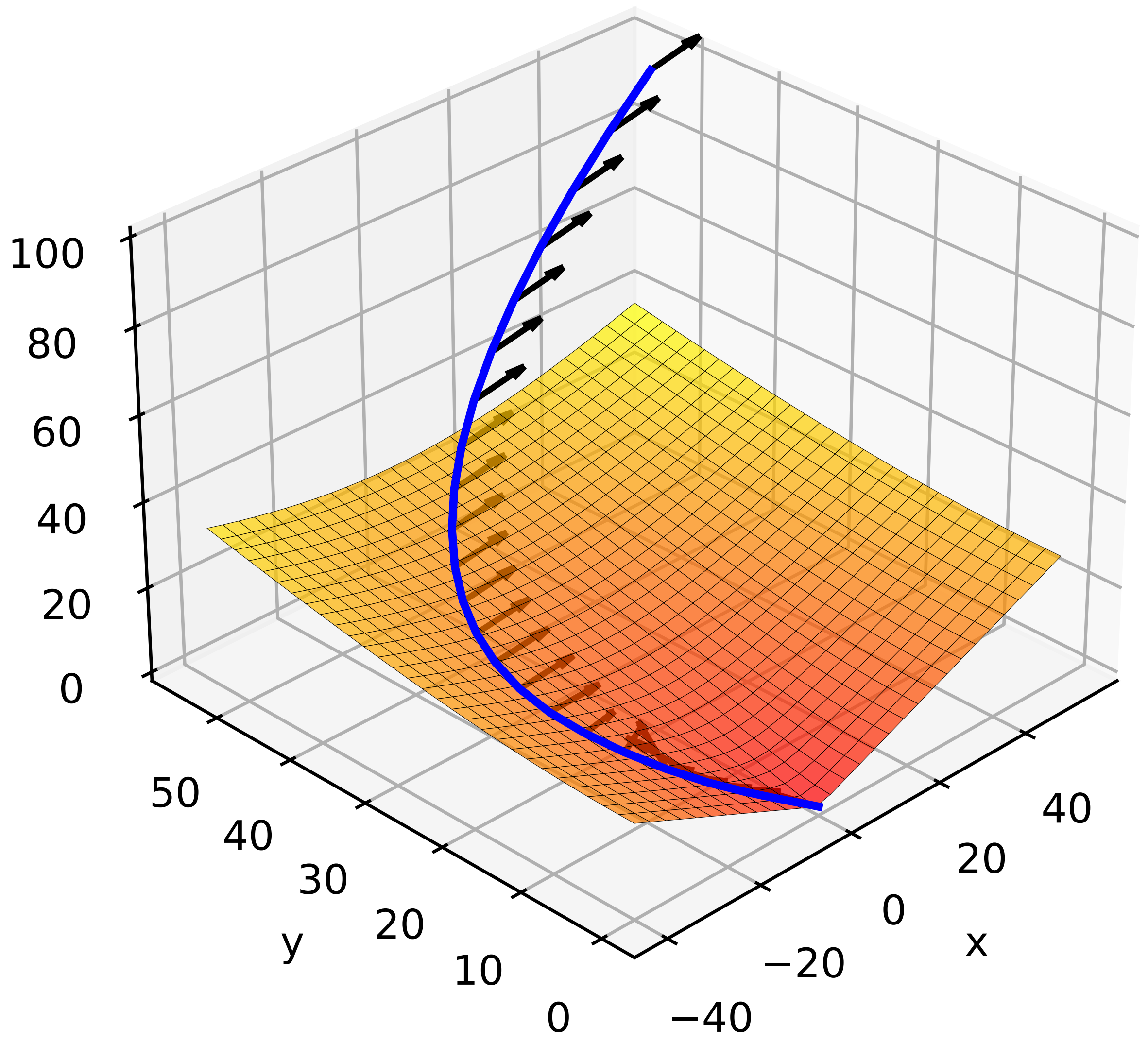

Figure 10.6: Minimum Touchdown Descent Trajectory and Thrust Profiles.

10.4 Model Predictive Control

10.4.1 Examples

Almost MPC

We apply MPC to the following output tracking example. The purpose of this exercise is simply to show how sequentially solving receeding horizon problems can converge to the prescient control problem (where the problem is solved with full access to the desired output) as the horizon is increased. Note that this example **does not** demonstrate the power of using MPC in a stochastic setting.

[Boy-8] **HW7 Q1.** *MPC for output tracking.* Consider the linear dynamical system

$$x_{t+1} = Ax_t + Bu_t, \quad y_t = Cx_t, \quad t = 0, \dots, T-1,$$

with state $x_t \in \mathbf{R}^n$, input $u_t \in \mathbf{R}^m$, and output $y_t \in \mathbf{R}^p$. The matrices A and B are known, and $x_0 = 0$. The goal is to choose the input sequence $u_1, \dots, u_t$ to minimize the output tracking cost

$$J_{\text{output}} = \sum_{t=1}^{T} \left\| y_t - y_t^{\text{des}} \right\|_2^2,$$

subject to $\|u_t\|_\infty \leq U^{\max}$, $t = 0, \dots, T-1$.

For the remainder of this problem we work with the specific problem instance with associated data

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0.5 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix},$$

$T = 100$, and $U^{\max} = 0.1$. The desired output trajectory is given by

$$y_t^{\text{des}} = \begin{cases} 0 & t < 30, \\ 10 & 30 \leq t < 70 \\ 0 & t \geq 70. \end{cases}$$

(a) Find the optimal input u^* and the associated optimal cost J^*.

Response. This is simply a *linear* (in the dynamics) *time-invariant quadratic tracking* problem. Instead of doing a theoretical analysis of controllability, etc., we can determine the feasibility of the control problem by formulating and attempting to solve the following convex optimization problem:

$$\begin{array}{ll} \text{minimize} & J_{\text{output}} = \sum_{t=1}^{100} \left\| Cx_t - y_t^{\text{des}} \right\|_2^2 \\ \text{subject to} & \|u_t\|_\infty \leq 0.1, \quad x_{t+1} = Ax_t + Bu_t, \quad t = 0, \dots, 99 \\ & x_0 = 0, \end{array}$$

(with the provided data for A, B, C, and y^{des}, of course.) Using CVXPY, we obtain the optimal cost $J^*_{\text{outout}} = 112.4157$.

(b) *Rolling look-ahead.* Now consider the input obtained using an MPC-like method where at time t, we find the values of $u_t, \dots, u_{t+N-1}$ that solve the following convex optimization problem

$$\begin{array}{ll} \text{minimize} & J_{\text{output}} = \sum_{\tau=t+1}^{t+N} \left\| C x_\tau - y_\tau^{\text{des}} \right\|_2^2 \\ \text{subject to} & \|u_\tau\|_\infty \leq 0.1, \quad x_{\tau+1} = A x_\tau + B u_\tau, \quad \tau = t, \dots, t+N-1 \\ & x_0 = 0. \end{array}$$

The value N is the amount of *look-ahead*, since it dictates how much of the future of the desired output signal we are allowed to access when we decide on the current input.
Find the input signal for look-ahead values $N = 8$, $N = 10$, and $N = 12$. Compare the cost J_{output} obtained in these three instances to the optimal cost J^*_{output} found in part (a).

Response. The Python code used to solve this problem can be found in these note's associated examples folder under 364b, so it is omitted here. The cost obtained by applying MPC with

- $N = 8$ is 379.634,
- $N = 10$ is 128.13,
- and $N = 12$ is 123.62.

Figure 10.7 shows the output trajectories for $N = 8$, $N = 10$, $N = 12$, the optimal output trajectory, and the desired output trajectory.

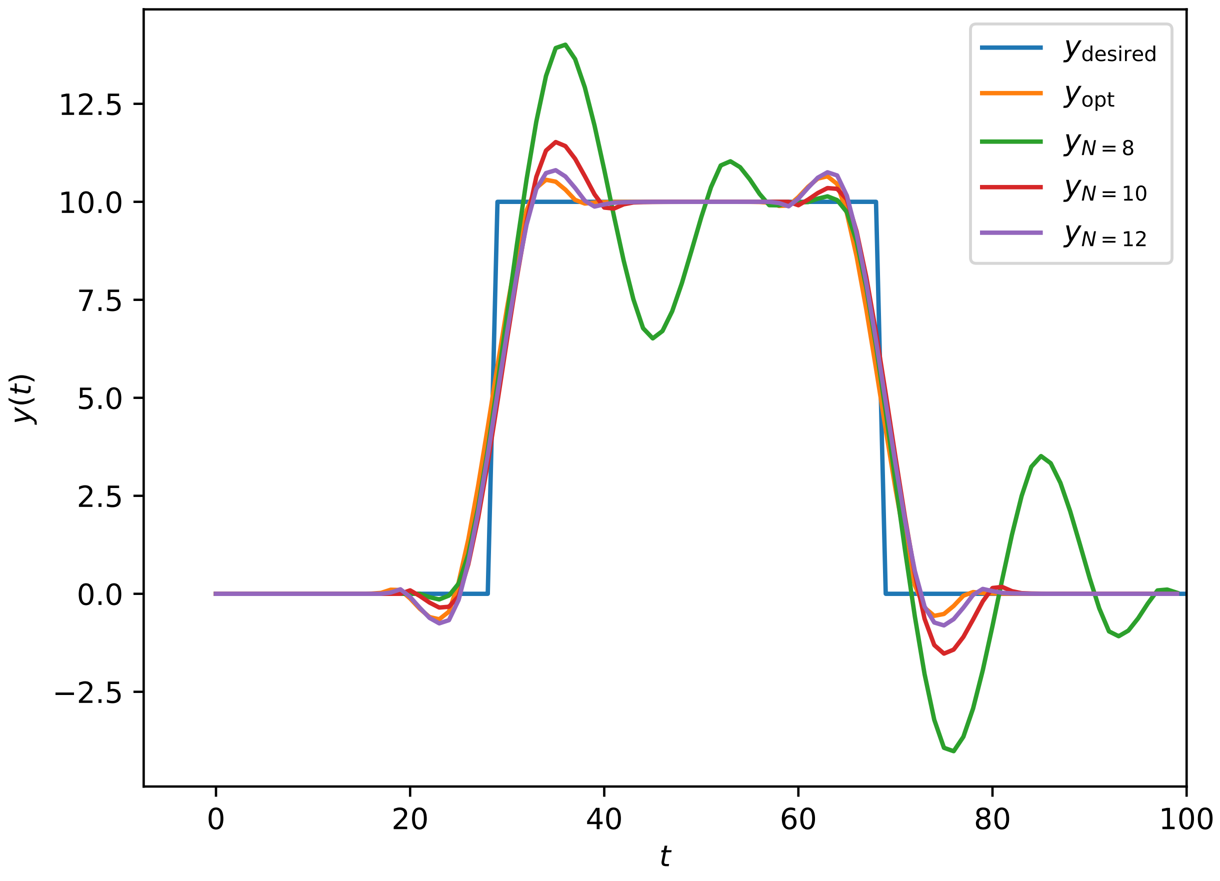

Figure 10.7: Output Trajectories.

Part IV

Algorithms

Chapter 11

Numerical Linear Algebra

11.1 QR Factorization

Bibliography

[Boy-8] Stephen Boyd. *EE36b: Convex Optimization II*. 2007-8. URL: https://see.stanford.edu/Course/EE364B.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004. ISBN: 978-0521833783.

[BV24] Stephen Boyd and Lieven Vandenberghe. *Additional Exercises for Convex Optimization*. 2024. URL: https://github.com/cvxgrp/cvxbook_additional_exercises.

[PG24] Marco Pavone and Daniele Gammelli. *AA203: Optimal and Learning-Based Control*. 2024. URL: https://stanfordasl.github.io/aa203/sp2324/.

[Rea20] RealPars. *What is an Actuator*. 2020. URL: https://www.youtube.com/watch?v=LHn7O6PUaoY.

Appendix A

Implementation

- For now setting is Python.
- Will include C++ gradually.
- Especially how to generate C optimization solvers from CVXPY.

A.1 CVXPY

- functools partial
- creating and unpacking list of tuples (and check if this is necessary)

A.1.1 Simulating (Linear) Dynamical Systems

A.2 C++