Case Studies

in

Linear Algebra and Optimization

Michael P. Friedlander
Nathan Krislock
Ting Kei Pong

Contents

Preface

Numerical linear algebra and optimization are wonderfully useful tools, but it is easy for students to lose sight of this while slogging through the demands of a typical course. The case studies in this book are designed to show students how the algorithms and theory that they are learning in class apply to compelling real-world situations. Our hope is that students working through these exercises will find motivation for further study.

Each of the eight case studies in this book features a particular aspect of numerical linear algebra or optimization within the context of a particular application. The case studies were designed for use in junior- and senior-level courses in these subjects in the Department of Computer Science at the University of British Columbia. They are meant to complement a regular course and textbook.

When we administered these case studies, we used the following format, which breaks into three parts, though we expect that other instructors will adapt these to suit the format of their particular course. The first part is meant to be done in class, and students work in small teams on simple exercises designed to help them understand the application and mathematical formulation, and the techniques that they should use to solve a problem. The exercises in the second part are assigned as individual homework, and generally involve writing code to implement an algorithm. In the final part, the students again assemble into teams and use their software implementations, together with data that we provide, to analyze the application. Students are asked to write a short report describing their approach and summarizing their solutions.

Students are not expected to build from scratch the code infrastructure needed for these case studies. Hyperlinks in the text lead to supporting code and data needed for each project. Students are asked to add functionality to the code provided, or to complete given code snippets. In most cases, we provide links to fully-working versions of the code, though these have been "obfuscated" to hide the solution. (The solutions are provided as MATLAB pcode functions, which are fully functional MATLAB scripts that are not human readable.)

The eight case studies, and the topics that they are meant to feature, are the following.

1. single-pixel camera: linear programming;
2. flow in a transportation network: network flow and sensitivity analysis;

3. Page rank: eigenvalue problems;
4. resistance in a social network: rank-one updates;
5. Poisson image editing: linear least-squares;
6. mouse tracking: block matrices and Cholesky factorization;
7. financial portfolio optimization: quadratic optimization;
8. image deblurring: regularization.

Funding needed for the development of these case studies was provided by the Carl Weiman Science Education Initiative.

Michael Friedlander
Nathan Krislock
Ting Kei Pong
December 2014

One

Single-Pixel Camera

Figure 1.1: Left: Original image of Leonhard Euler (1707-1783). Right: A 20-times compressed image using the wavelet basis.

1 Learning goals

This case study is an example of a modern application in signal processing concerning the recovery of signals (in this case, an image) from incomplete measurements. The recovery procedure is based on computing a minimal-norm solution to an underdetermined system of linear equations. We expect that students will be familiar with the idea of "solving" an underdetermined linear system—e.g., by using the right pseudo-inverse of the matrix. In the solution procedure used here, however, there are two twists that may be unfamiliar to students. The first, is that a minimal 1-norm solution is required, and the pseudo-inverse solution from classical linear algebra doesn't apply. Instead, we must solve a problem closely related to linear programming. The second twist is that the matrices are large, and are more practically represented as linear operators for which fast algorithms are available.

2 Background

We will use methods from *compressed sensing* to build a virtual single-pixel camera. By taking advantage of the sparsity of an image in some basis, we can take a picture using just a single-pixel sensor. In a single-pixel camera, the light from the image is reflected off an array of tiny mirrors, which then passes through a lens that focuses the light onto a single photosensor that measures the overall light intensity reflected off a subset of the mirrors.

Clearly, we will need to take more than one measurement of the image with a single-pixel sensor in order to recover the image. It's also clear that we could take a measurement for each pixel in the image by measuring each individual pixel separately. However, because of sparsity, we will be able to create a good-quality picture using much fewer measurements than the total number of pixels in the image.

3 Problem description

Suppose the original image is given by an $m \times n$ matrix X. In the wavelet basis, the image X is sparse. (You will verify this property experimentally). Let Q be an $mn \times mn$ matrix that represents the wavelet basis. Then the wavelet coefficients y of X are given by $y = Q\text{vec}(X)$, where the mn-vector $x := \text{vec}(X)$ is obtained by stacking the columns of the matrix X. Because Q is invertible, $\text{vec}(X) = Q^{-1}y$.

Each "measurement" b_i of the image—given by a particular configuration of the reflecting mirror array—can be modeled by the inner product $b_i := m_i^T x$. Stacking the measurements $i = 1, \ldots, q$ into a vector, we can represent the q measurements as the product

$$b = \begin{bmatrix} m_1^T x \\ \vdots \\ m_q^T x \end{bmatrix} = Mx, \quad \text{where} \quad M := \begin{bmatrix} m_1^T \\ \vdots \\ m_q^T \end{bmatrix}$$

is a $q \times mn$ matrix with 0/1 entries. Each entry in M corresponds to an active or inactive mirror in the array. We expect the coefficients y of the image X in the basis given by Q to be very sparse, so we can recover y by solving the *basis pursuit* problem

$$\underset{y}{\text{minimize}} \quad \|y\|_1 \quad \text{subject to} \quad Ay = b,$$

where the $q \times nm$ matrix A depends on M and Q. (The dependence of A on M and Q will be explored as in the exercise of Part 2. Moreover, the number q of random measurements can be much smaller than the number of pixels in the image X and is typically a small multiple of the number nonzero coefficients of y.

4 Exercises

You will need to install two MATLAB toolboxes: Spot and SPGL1.

4.1 (*In class*). In MATLAB, load and display `Euler256.png` using the following commands:

```
X = double(imread('Euler256.png'));
imagesc(X);  colormap gray;  axis square;
```

Compress the image stored in the matrix X using the *discrete cosine transform* basis D, the *Haar wavelet* basis H, and the *Daubechies wavelet* basis W. Although these matrices are large and dense, we can use the Spot linear-operator toolbox to efficiently work with them in MATLAB:

```
[m, n] = size(X);
D = opDCT2(m, n);
H = opHaar2(m, n);
W = opWavelet2(m, n);
```

Complete the MATLAB function `compress.m`:

```
function Xcompress = compress(X, Q, p)
%COMPRESS Compress image using an operator.
%
% Xcompress = COMPRESS(X, Q, p) compresses the image
% in X and returns the resulting image Xcompress.
%
% Q is the operator describing the basis to be used
% for the compression.
% p is the compression ratio (0 <= p <= 1).
```

Use compression ratios of $p = 10\%$, 5%, and 2%, and display the compressed images using each of the three operators, D, H, and W. You may compare your output against the solution `compress_soln.p`.

4.2 (*Homework*). Download the MATLAB function `measure.m`:

```
function [b, M] = measure(X, q)
%MEASURE Make q random measurements of an image.
%
% [b, M] = measure(X, q) compute q random measurements
% of the image X and returns the measurement operator
% M and b = M*X(:).
```

The $q \times mn$ operator M represents a *single-pixel camera* that takes q random measurements of the $m \times n$ image X using a single-pixel sensor.

Download and complete the MATLAB function `reconstruct.m`:

```
function X = reconstruct(b, M, Q, m, n)
%RECONSTRUCT Reconstructs an image from measurements.
%
% X = RECONSTRUCT(b, M, Q, m, n) reconstructs the
```

```
% m-by-n image X from the measurements b taken from
% the true image using the measurement operator M.
%
% The image X is reconstructed from the measurements b
% by solving the basis pursuit optimization problem
% for the coefficients y of X in the basis given by
% the operator Q:
%
%     minimize   ||y||_1   subject to   Ay = b,
%
% where A is an operator that depends on the operators
% M and Q.
%
% The optimal solution y is found using the spg_bp
% routine of the SPGL1 sparse reconstruction solver:
%
%     http://www.cs.ubc.ca/~mpf/spgl1/
```

Run the following with different values of q, using the three possible operators Q:

```
[b, M] = measure(X, q);
Xrc = reconstruct(b, M, Q, m, n);
```

You may compare your output against the solution `reconstruct_soln.p`.

4.3 (*In class).* The theory of *compressed sensing* asserts that, with high probability, we can recover a sparse signal y by solving the *basis pursuit* optimization problem:

$$\underset{y}{\text{minimize}} \quad \|y\|_1 \quad \text{subject to} \quad Ay = b,$$

where the number of measurements b_i, $i = 1, \ldots, q$, is some small multiple of the number of nonzero entries in the signal y. Suppose the number of nonzero entries of y is k.

```
Q = W;  p = .10;
Xcompress = compress(X, Q, p);
y = Q*Xcompress(:);
k = nnz(round(y)); % k = number of nonzeros of y
```

Using the Daubechies wavelet basis (`Q = W`), perform a computational experiment to determine the value of $i = 1, 2, \ldots$ for which $q = ik$ measurements of `Xcompress` are sufficient to recover y. That is, find the smallest value of i for which `Xrc` is visually similar to `Xcompress` when compared side-by-side.

```
subplot(1,2,1); imagesc(Xcompress);
colormap gray; axis square;
[b, M] = measure(Xcompress, i*k);
Xrc = reconstruct(b, M, Q, m, n);
subplot(1,2,2); imagesc(Xrc);
```

```
    colormap gray; axis square;
```

Repeat the same experiment when measuring `X` and comparing the recovered image `Xrc` with the compressed image `Xcompress`. For what value of i does the person in the image `Xrc` become recognizable?

Two

Flow in a Transportation Network

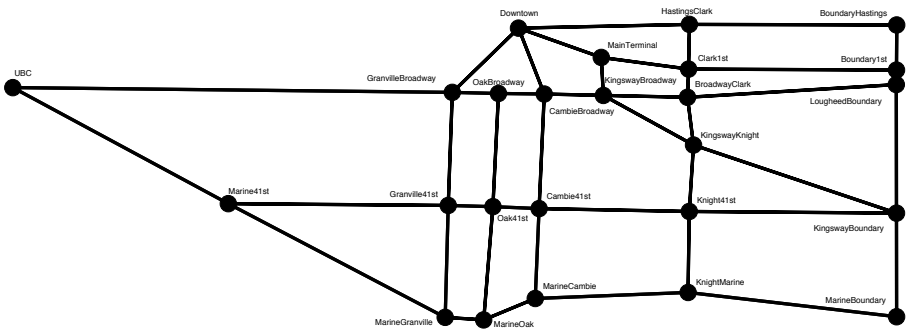


Figure 2.1: The major road network of Vancouver.

1 Learning goals

In this case-study, we will learn how to create and solve large linear programming models of the following network flow problems:

- max-flow (determining the maximum possible flow from a source node to a sink node subject to capacity constraints);
- min-cost flow (determining the minimum cost way to move goods from supply nodes to destination nodes);
- min-cost multi-commodity flow (same as the min-cost flow problem, but with more than one type of commodity to transport).

We will study these problems in the context of the public transportation system in Vancouver where many passengers travel into Vancouver everyday to go to work in the downtown area or to go to study at the University of British Columbia. In addition to learning practical methods to formulate and solve these problems, we will also see how to find the most effective ways to improve the network by studying the optimal dual multipliers.

We will formulate these models in the GNU Mathematical Programming Language (GMPL) and solve them using the GNU Linear Programming Kit (GLPK). In addition, we use MATLAB to interact with the GLPK solver and to visualize the optimal flows on the network.

We expect the students will be familiar with solving small linear programming problems by hand using the simplex method, but may not have had much experience with using mathematical software for solving large linear programming problems. We also assume that students have some familiarity with linear programming duality and dual multipliers.

2 Background

We will look at a variety of different types of *network flow problems* to analyze public transit in Vancouver. The network flow problems we study here are linear programs which can be very efficiently solved using the simplex method.

Let $G = (V, E)$ be a *directed graph* with node set V and edge set $E \subseteq V \times V$. We will let $x_{ij} \geq 0$ represent the amount of *flow* (e.g., number of people per hour) from node i to node j along the edge (i, j). We can write the total amount of flow leaving node i as $\sum_{j:(i,j)\in E} x_{ij}$, and the amount of flow arriving at node i as $\sum_{j:(j,i)\in E} x_{ji}$. We let

$$\delta_x(i) := \sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij}$$

represent the *net flow* at node i.

A basic constraint in many network flow problems is that the amount of flow going into a node must be equal to the amount of flow leaving that node. We can write this *flow conservation* constraint at node i as $\delta_x(i) = 0$. In general, we have flow *demand* constraints of the form

$$\delta_x(i) = d_i, \quad \text{for } i \in V,$$

where the demand d_i may be a constant (positive, negative, or zero) or a variable. Note that a negative demand is used to indicate a *supply*.

3 Problem description

As mentioned above, we will study three types of network flow problems.

1. In the maximum flow problem, the aim is to determine the maximum possible flow from a *source* node $s \in V$ to a *sink* node $t \in V$, subject to capacity constraints on the edges. We can write the maximum flow

problem as

$$\begin{array}{lll} \underset{x,f}{\text{maximize}} & f & \\ \text{subject to} & \delta_x(s) = -f, & \\ & \delta_x(t) = f, & \\ & \delta_x(i) = 0, & i \in V \setminus \{s,t\}, \\ & 0 \le x_{ij} \le c_{ij}, & (i,j) \in E, \\ & f \ge 0. & \end{array}$$

where c_{ij} is the capacity of edge (i,j). Since all the constraints are linear functions of the variables x and f, the maximum flow problem is a linear program.

We will also consider the maximum flow problem having a set S of source nodes and a set T of sink nodes. We can formulate this problem as

$$\begin{array}{lll} \underset{x,f}{\text{maximize}} & f & \\ \text{subject to} & \displaystyle\sum_{s \in S} \delta_x(s) = -f, & \\ & \displaystyle\sum_{t \in T} \delta_x(t) = f, & \\ & \delta_x(i) = 0, & i \in V \setminus (S \cup T), \\ & 0 \le x_{ij} \le c_{ij}, & (i,j) \in E, \\ & f \ge 0. & \end{array}$$

2. In the minimum-cost flow problem, we have a cost m_{ij} per unit flow for each edge $(i,j) \in E$, and demands d_i for each node $i \in V$. We want to find the flow with minimum total cost that satisfies the demand and capacity constraints. The problem can be formulated as

$$\begin{array}{lll} \underset{x}{\text{minimize}} & \displaystyle\sum_{(i,j) \in E} m_{ij} x_{ij} & \\ \text{subject to} & \delta_x(i) = d_i, & i \in V, \\ & 0 \le x_{ij} \le c_{ij}, & (i,j) \in E. \end{array}$$

   Note that it is necessary that the demands satisfy $\sum_{i \in V} d_i = 0$.

3. The minimum-cost multi-commodity flow problem is similar to the previous problem except that we now have a set of commodities K. We define x_{ij}^k to be the flow of commodity $k \in K$ over edge $(i,j) \in E$. In addition, we let d_i^k be the demand for commodity $k \in K$ at node $i \in V$. The different commodity flows must each satisfy their own demand constraints. However, the different flows must share the network in the sense that

together they must not exceed the capacity of the edges. We can state this problem as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{(i,j)\in E} m_{ij} \sum_{k\in K} x_{ij}^k \\ \text{subject to} \quad & \delta_{x^k}(i) = d_i^k, && k \in K,\ i \in V, \\ & \sum_{k\in K} x_{ij}^k \le c_{ij}, && (i,j) \in E, \\ & x_{ij}^k \ge 0, && k \in K,\ (i,j) \in E. \end{aligned}$$

As before, it is necessary that the demands satisfy $\sum_{i\in V} d_i^k = 0$, for all $k \in K$.

4 Exercises

Software prerequisites. You will need to install GLPK (GNU Linear Programming Kit).

4.1 (*In class 1).*

1. Install GLPK (GNU Linear Programming Kit) on the machine which you use to run MATLAB. Make sure you can run the GLPK command `glpsol` in MATLAB:

```
>> !glpsol
GLPSOL: GLPK LP/MIP Solver, v4.48
No input problem file specified; try glpsol --help
```

2. Download the GMPL (GNU MathProg Language) model file `maxflow.mod` and data file `maxflow.dat`. Download the `solve_network_flow.m` file and use it to solve the maximum flow problem as follows:

```
>> prob.model = 'maxflow.mod';
>> prob.data = 'maxflow.dat';
>> solution = solve_network_flow(prob);
```

This finds the maximum flow from `KingswayBoundary` to `UBC`. Download `plot_flow.m` and use it to display the solution.

3. A *bottleneck* is a set of edges that are collectively restricting the flow in the sense that increasing the capacity of any of these edges will increase the maximum possible flow. Identify the bottleneck in the network.

4. Solve the max flow problem again by directly calling `glpsol`, and ask it to generate a solution file `maxflow.sol`:

```
>> !glpsol -m maxflow.mod -d maxflow.dat -o maxflow.
   sol
```

The column in `maxflow.sol` that is labelled `Marginal` represents the dual multipliers for the constraints. The marginal value of each edge gives the rate of change of the optimal value obtained by increasing or decreasing the capacity of that edge. Which edges have the largest marginal values? How is this related to the bottleneck you identified? Comment on your results.

5. Create `maxflow_multi.mod` and `maxflow_multi.dat` to find the maximum total flow from the following set of nine source nodes

```
set SOURCES :=
MarineGranville  MarineOak      MarineCambie
KnightMarine     MarineBoundary KingswayBoundary
LougheedBoundary Boundary1st    BoundaryHastings ;
```

to the following set of two sink nodes

```
set SINKS := Downtown UBC ;
```

subject to the same capacity constraints as before. Solve the problem and display the flow using `plot_flow.m`. Comment on your results.

6. Create a new data file `maxflow_skytrain.dat` which adds the Canada Line (CL), with a frequency of 12 trips per hour and a capacity of 200 passengers per trip, and the Expo/Millennium Line (EML), with a frequency of 24 trips per hour and a capacity of 200 passengers per trip. Use the following routes for the Canada Line and the Expo/Millennium Line.

```
set ROUTE[CL] :=
        MarineCambie          Cambie41st
          Cambie41st      CambieBroadway
      CambieBroadway            Downtown ;

set ROUTE[EML] :=
    KingswayBoundary      KingswayKnight
      KingswayKnight    KingswayBroadway
    KingswayBroadway        MainTerminal
        MainTerminal            Downtown ;
```

Solve the problem and display the flow using `plot_flow.m`. Comment on your results.

4.2 (*Homework).* We will now find the *minimum cost flow* that satisfies certain supply/demand constraints on the nodes and capacity constraints on the edges, using the costs described below. Start by creating copies of

`maxflow.mod` and `maxflow_skytrain.dat`, and naming them `mincost.mod` and `mincost.dat`.

1. Suppose it costs \$0.10 to transport one passenger one kilometre. Add the following to your `mincost.dat` file:

```
param unitcost := .10;  # cost per passenger per km
```

In your `mincost.mod` file, use the haversine formula to compute the length of each edge in kilometres from the longitude/latitude coordinates of its endpoints.

```
param pi := 3.14159265358979;
param R := 6371;   # mean radius of the Earth in km

param a{(i,j) in EDGES} :=
    sin(pi*( (coord[i,'lat'] - coord[j,'lat'])/2 )
    /180)^2 +
    (
    cos(pi*coord[i,'lat']/180) *
    cos(pi*coord[j,'lat']/180) *
    sin(pi*( (coord[i,'long'] - coord[j,'long'])/2 )
    /180)^2
    );

param dist{(i,j) in EDGES} :=
        2*R*atan( sqrt(a[i,j]), sqrt(1-a[i,j]) );
```

Use the length of each edge to compute the cost of transporting one passenger along that edge, where you should define cost in your `mincost.mod` file as:

```
param cost{(i,j) in EDGES} := unitcost * dist[i,j];
```

We want to minimize the total cost per hour:

```
minimize total_cost:
    sum{(i,j) in EDGES} cost[i,j] * Flow[i,j];
```

2. Set supply/demand constraints as follows. Put

```
param demand {NODES} default 0;
```

in your `mincost.mod` file and put

```
param demand :=
     MarineGranville            -500
           MarineOak            -500
       MarineCambie             -500
    KingswayBoundary            -500
```

```
        BoundaryHastings            -500
                     UBC            2500 ;
```

in your `mincost.dat` file. All other parameters are as in your `maxflow_skytrain.dat` file. Solve the problem and display the flow using `plot_flow.m`. Comment on your results.

3. The dual multiplier for each edge gives the rate of change of the minimum cost per unit increase or decrease in the capacity of the edge. After solving the problem in the model file, the value of the dual multiplier is stored in `Flow[i,j].dual`. Which edge has the most negative dual multiplier? Based on this, how would you recommend the transit network be improved to reduce the operating cost required to meet the above demands? Create a new data file, `mincost2.dat`, that reflects your proposed improvement and find the new operating cost.

4.3 (*In class 2).* We will now consider separate flows for travellers going to UBC and travellers going downtown. Such a problem is known as a *minimum cost multi-commodity flow* problem. Start by creating copies of `mincost.mod` and `mincost.dat`, and naming them `multicommodity.mod` and `multicommodity.dat`.

1. Given a set of destinations, we add a destination index to the `Flow` variable. Put the following into your `multicommodity.mod` file.

```
set DESTINATIONS within NODES;
var Flow{EDGES, DESTINATIONS} >=0;
```

We now have separate demands for each type of flow. Put the following into your `multicommodity.dat` file.

```
set DESTINATIONS := Downtown UBC ;

param demand :            Downtown         UBC :=
    KingswayBoundary         -3000       -2000
        MarineCambie         -1000        -500
     MarineGranville          -500        -500
           MarineOak          -500       -1000
            Downtown          5000           0
                 UBC             0        4000 ;
```

Update the objective function and demand constraints appropriately, and do not forget to include capacity constraints. Solve the problem and display the flow using `plot_flow.m`. Comment on your results.

2. Analyze the solution using the dual multipliers and propose an improvement to the network.

Three

Page Rank

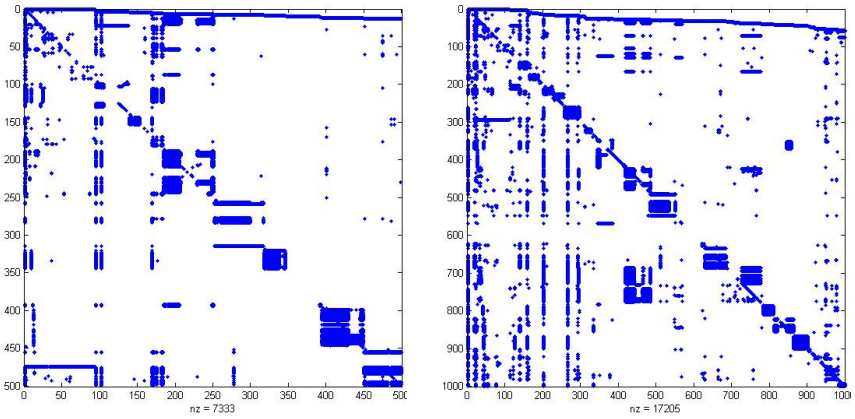


Figure 3.1: Hyperlink matrices.

1 Learning goals

This case study takes a look at the basic mechanism behind Google searches: the so-called PageRank algorithm. We expect that students will be familiar with the idea of left and right eigenvalues and eigenvectors of a matrix, and also some basic procedures for finding eigenvalues for small matrices. In our discussion here, there are two twists that may be new to the students. First, the data matrix is huge and sparse, rather than small and dense. In the solution procedures, the students must use only matrix-vector multiplications and avoid matrix-matrix multiplications at all costs. Students are strongly encouraged to compare their codes with the solution codes in terms of speed to make sure that their implementations are efficient. Second, due to the huge matrix size, an iterative method, the power method, will have to be used in order to find the largest eigenvalue of the matrix. The convergence of the power method in this case is a consequence of the Peron Fröbenius theorem. This latter theorem, though important, is typically not discussed in introductory linear algebra courses due to time constraints.

2 Background

Search engines such as Google return a list of webpages based on their relevance to a query. Such a ranking is usually based on the computation of two different scores: a content score that is query dependent, and a popularity score that is merely dependent on the Internet network structure. In this case study, we look at the mathematics behind the popularity score: the PageRank algorithm. The algorithm consists of finding an eigenvector corresponding to the largest eigenvalue of a matrix suitably constructed out of the hyperlink structure.

3 Problem description

In the PageRank algorithm, (part of) the Internet is modeled using a (nonsymmetric) $n \times n$ matrix H, where the rows and columns are indexed by webpages and H_{ij} is nonzero if there is a hyperlink on webpage i leading to webpage j. All entries of H are nonnegative, with each row summing to 1, so that H_{ij} can be interpreted as the transition probability to webpage j when the surfer is on webpage i. Following the thesis that "a page is important if it is pointed to by other important pages", a first definition of pagerank π_j of a webpage j would be a weighted sum of the pageranks of other pages. Mathematically, $\pi^T = \pi^T H$. We would also require π to be a probability distribution, i.e., $\sum \pi = 1$, $\pi \geq 0$.

Two modifications are further made to the above model to guarantee existence of the pagerank vector π. First, we form $S = H + ae^T/n$, where a is the vector whose ith entry is 1 when the ith row of H is zero, and is zero otherwise; and e is the vector of all ones. Zero rows in H correspond to webpages that contain no hyperlinks to other pages. After the above modification, the corresponding rows in S become e^T/n, meaning that a surfer arriving at those webpages will go to another page at random. Second, we pick any $\alpha \in (0, 1)$ and set $G = \alpha S + (1-\alpha)ee^T/n$. This matrix G is called the Google matrix, and α controls how often a surfer actually follows the hyperlink structure coded in S (rather than browsing completely randomly).

The problem of ranking webpages is to find π so that

$$\pi^T = \pi^T G, \quad \sum \pi = 1. \tag{3.1}$$

Webpages are then ranked according to the magnitude of the entries of π. Since G is a stochastic matrix with positive entries, by the Perron-Fröbenius theorem, such a π exists and is unique, with positive entries. Indeed, it is a "normalized" left eigenvector of the Google matrix corresponding to the largest eigenvalue in magnitude, 1.

3.1 (*In class*).

1. Download the MATLAB MAT file `WebUBC1.mat`. It contains two variables, U and H. The variable U is a cell array consisting of web addresses. The variable H is the matrix described in the introduction, with rows and

columns indexed by the webpages in U. The information was constructed using the hyperlink structure in UBC webpage, taking into account a total of 500 webpages it points to, obtained via crawling.

Use the following command to load and display the sparsity pattern of the matrix H.

```
load WebUBC1
spy(H)
```

2. Write a MATLAB function

```
function G = Gmatrix(H,alpha)
% This outputs the Google matrix
%
% G = alpha*S + (1-alpha)*e*e'/n
%
% where S = H + a*e'/n, a_i is 1 if the ith row of H
% is zero, and is 0 otherwise.
```

You may compare your output against Gmatrix_soln.p.

3. Download and complete the M-file Pagerank_one.m.

```
Pagerank = Pagerank_one(G)
```

This function takes in the Google matrix G, and computes the pagerank vector, i.e., the solution of the eigenvalue problem (3.1). You may want to look up `eig` in MATLAB and compare your output against Pagerank_one_soln.p.

Take $\alpha = 0.85$, a value suggested by the founders of Google. Which index corresponds to the largest pagerank value? Which webpage does it correspond to? (`U{1}` gives the first entry of the cell array) Do you think the result is reasonable?

Plot the index corresponding to the largest pagerank value against $\alpha = 0.05, \ldots, 0.95$. Identify the corresponding webpages.

3.2 (*Homework).* In real world applications, the size of the Google matrix G is huge: just imagine how many webpages there are. It is thus unrealistic to find the pagerank vector using eigenvalue decompositions as in the MATLAB function `eig`. One alternative is to use an iterative method.

Notice that we only need the eigenvector corresponding to the eigenvalue of largest magnitude. This can be conveniently found by the power method. In short, this method is initialized at a positive vector π_0. In each iteration, we update

$$\pi_{k+1} = \frac{\pi_k^T G}{\pi_k^T G e},$$

where e is the vector of all ones. The algorithm can be terminated when the change in successive iterates is small.

1. Download and complete `Pagerank_power.m`.

   ```
   Pagerank = Pagerank_power(H,alpha,tol)
   ```

   This function takes in the hyperlink matrix H, the parameter α for the construction of the Google matrix, and a parameter `tol` for the termination of the power method. It computes an eigenvector corresponding to the eigenvalue of largest magnitude of the Google matrix G, and outputs the pagerank vector. (Hint: You do not need to form the Google matrix explicitly.)

   You may want to compare your output against `Pagerank_power_soln.p` on the hyperlink matrices given in Exercise 1. Set `tol=1e-5` in your tests.

2. We now apply the above code to a huge network. Make sure your code does not involve explicitly forming the Google matrix!

   Download the MATLAB MAT file `Textmatrix.mat`, which contains a 281903×281903 hyperlink matrix describing webpage links in Stanford network. Plot the index corresponding to the largest pagerank value against $\alpha = 0.05, \ldots, 0.95$.

3.3 (*In class).* As an attempt to "personalize" search preferences, a *personalization vector* w was introduced. Specifically, we modify the definition of Google matrix as follows:

$$G = \alpha S + (1 - \alpha) e w^T \tag{3.2}$$

for some probability vector w. In this setting, the browser either follows the hyperlink structure coded in S, or randomly browse/teleport according to the probability vector w. When the probability is uniform, i.e., $w = e/n$, this recovers the definition of Google matrix in the introduction.

1. Based on your code `Pagerank_power.m`, write a MATLAB code that takes in the hyperlink matrix H, the parameter α for the construction of the Google matrix, the (column) personalization vector w and a parameter `tol` for the termination of the power method. Compare your output against `Pagerank_power_personal_soln.p`.

   ```
   Pagerank = Pagerank_power_personal(H,alpha,w,tol)
   ```

2. Download `yahoomatrix.mat`, which records the hyperlink data obtained by crawling `http://travel.yahoo.com`.

   a) Take $w = e/n$. Plot the index corresponding to the largest pagerank value against $\alpha = 0.05, \ldots, 0.95$ and identify the corresponding webpages.

   b) We now test the idea of the personalized search. Use the following commands to extract the indices that contain the characters `flickr`.

```
Indexsave = [];
for ii = 1:size(U,1)
  if findstr('flickr',U{ii});
    Indexsave = [Indexsave; ii];
  end
end
```

The indices of the webpages in U that contains the characters `flickr` are stored in the vector `Indexsave`.

Construct a w so that it is zero outside `Indexsave`, and has the same value over all indices in `Indexsave`. Plot the index corresponding to the largest pagerank value against $\alpha = 0.05, \ldots, 0.95$ and identify the corresponding webpages. How does this compare with the result of part (a)?

Four

Resistance in a Social Network

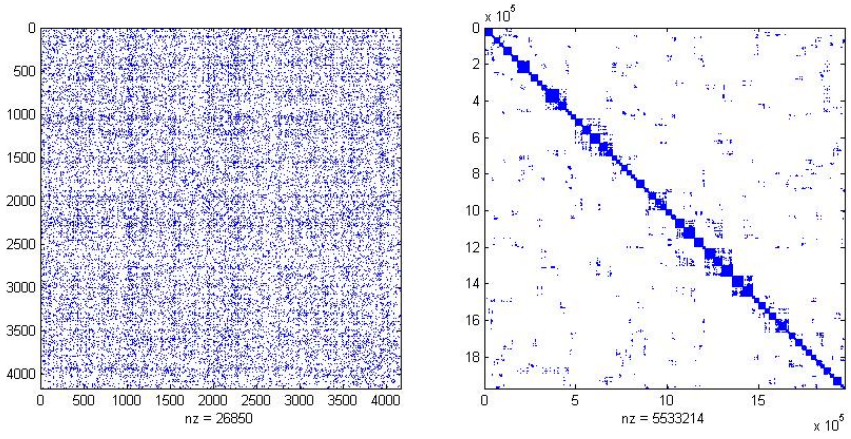


Figure 4.1: Collaboration (left) and traffic (right) networks.

1 Learning goal

This case study concerns an important question in social networks: how closely related are two people in the network. The notion of proximity we use is the resistance distance. It is demonstrated how various rank-one update techniques can help compute the resistance distances efficiently upon small changes in the network. We focus on the Cholesky rank-one updates, and will also briefly discuss the Sherman-Morrison formula in the last exercise. We expect that students will be familiar with the idea of the Cholesky factorization of positive definite matrices. In this case study, the students will be guided through deriving the formula for the Cholesky rank-one update. However, there are two points that may be new for the students. First, they will have to figure out the vector used for the rank-one update when one additional connection in the network is identified. Second, when matrices are large, the students will observe that there can be a big difference in the computational time when the compu-

tation involves matrix-matrix multiplications rather than only matrix-vector multiplications.

2 Background

One interesting question in a social network is how "close" are two (groups of) people. The notion of closeness makes this question tricky. Take the network structure induced by Facebook friendship as an example. Such a structure can be conveniently modeled using an undirected graph, with a node representing an individual, and an edge between them if they are friends on Facebook. A measure of closeness could be the length of the shortest path between the two people. However, this notion does not capture the intuition that two persons should be closer if they do not only know each other but also have a lot of common friends. In this case study, we will look at a recently proposed notion of distance that captures this intuition, the resistance distance.

3 Problem description

Let A be the adjacency matrix corresponding to a social network $G = (V, E)$, i.e., A is a symmetric $n \times n$ matrix whose rows and columns are indexed by the nodes in V, and that $A_{ij} = 1$ if $(i, j) \in E$, and $A_{ij} = 0$ otherwise. The Laplacian matrix L is defined as

$$L := \text{Diag}(d) - A = \sum_{(i,j)\in E} (e_i - e_j)(e_i - e_j)^T,$$

where $\text{Diag}(d)$ is the diagonal matrix whose ith diagonal entry is the degree of node i, and e_i is the vector whose ith entry is one, and is zero otherwise. The Laplacian is positive semidefinite. For connected graphs, its nullspace is spanned by e, the vector of all ones.

The resistance distance between nodes i and j on a connected graph is defined as

$$r_{ij} := (e_i - e_j)^T (L + ee^T)^{-1} (e_i - e_j).$$

It can be shown that the above quantity remains the same if $L + ee^T$ is replaced by $L + \alpha ee^T$ with $\alpha > 0$. This notion of distance has its origin from electrical networks. In that context, it computes the resistance between the two points i and j in the network, assuming unit resistance on each edge. Its relationship with the information distance on social network was recently identified.

4 Exercises

4.1 (*In class).*

1. Download the matrix file `Graph.mat` (taken and modified from the UF Sparse Matrix Database). This matrix is an adjacency matrix of a collaboration network on "General Relativity" on arXiv: node i is connected to node j if they are coauthors. Use the following commands to load and view the matrix.

```
load Graph
spy(A);
```

2. Let $L + ee^T = R^T R$ be the Cholesky factorization of $L + ee^T$, where R is upper triangular. Show that

$$r_{ij} = \|(R^T)^{-1}(e_i - e_j)\|^2.$$

Based on this, write a MATLAB function to compute r_{ij} for any given i and j.

```
[r,R] = Resistance(Lones,i,j);
```

the inputs are the positive definite matrix `Lones` (corresponding to $L + ee^T$ in the problem), i and j. The $r = r_{ij}$ is the resistance distance, and R is the Cholesky factor of `Lones`.

You may compare your output against `Resistance_soln.p`.

3. Download and complete `changeResistance.m`.

```
r = changeResistance(A,i,j,k);
```

This code takes in the adjacency matrix A, i and j, and also a positive integer k. It connects i to at most k random neighbors of j one after one, and recomputes the resulting resistance distance between i and j after each addition. All the computed resistances are recorded in the output vector r.

Experiment your code with $i = 500$, $j = 3000$ and $k = 10$. You may compare your output against `changeResistance_soln.p`. You can use `rng('default')` to reset the random seed for comparing the two codes.

Observe the decrease in the resistance distance as k increases. This indicates, intuitively, if researcher i works with coauthors of researcher j, they get "closer" to each other on this collaboration network.

4.2 (*Homework).* Notice that a Cholesky factorization is performed each time the resistance distance is computed. The overall computation can be sped up using a Cholesky rank-one update. This exercise is devoted to developing a MATLAB code for performing a Choleksy rank-one update.

Suppose that B is an $n \times n$ positive definite matrix and $B = R^T R$ is the Cholesky factorization. Given this R and a vector x, we would like to obtain a

Cholesky factorization for $B + xx^T$. To describe the rank one update, we first note that

$$B + xx^T = R^T R + xx^T = \begin{bmatrix} R^T x \end{bmatrix} \begin{bmatrix} R \\ x^T \end{bmatrix}.$$

The matrix $\begin{bmatrix} R \\ x^T \end{bmatrix}$ is $(n+1) \times n$.

1. Let $(r_0, x_0) \in \mathbb{R}^2$ be given with $r_0 > 0$. Find (c, s) with $c > 0$ and $c^2 + s^2 = 1$ so that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{pmatrix} r_0 \\ x_0 \end{pmatrix} = \begin{pmatrix} \sqrt{r_0^2 + x_0^2} \\ 0 \end{pmatrix}. \tag{4.1}$$

   The matrix is an example of a Givens rotation matrix. This class of matrices is orthogonal.

2. One way to perform a Cholesky rank-one update is to apply successive Givens rotation matrices Q to "zero out" the x step by step. We choose a suitable Givens matrix

$$Q_1 = \begin{bmatrix} c & 0 & -s \\ 0 & I & 0 \\ s & 0 & c \end{bmatrix}$$

   so that

$$Q_1 \begin{bmatrix} R \\ x^T \end{bmatrix} = Q_1 \begin{bmatrix} r_0 & \bar{r}^T \\ 0 & R' \\ x_0 & \bar{x}^T \end{bmatrix} = \begin{bmatrix} \sqrt{r_0^2 + x_0^2} & \bar{r}'^T \\ 0 & R' \\ 0 & \bar{x}'^T \end{bmatrix} =: R_1.$$

   Notice that only the first and last rows are modified in the output, due to the position of I in Q_1. Verify that Q_1 is orthogonal and that $R_1^T R_1 = B + xx^T$. A matrix Q_2 is then similarly chosen to operate on the second and last rows of R_1 so as to zero out the first entry of $\bar{x}'$, and so on.

   Download and complete `mycholupdate.m`, which takes the above approach to compute Cholesky factor of $B + xx^T$, given R and x.

```
R = mycholupdate(R,x);
```

   Remember, you do *not* need to form the whole Givens matrix; you only need c and s. Test your code with random instances:

```
B = randn(3000); B = B'*B;
R = chol(B); x = randn(3000,1);
```

   You may compare your code against `mycholupdate_soln.p`. After that, you may also take a look at the MATLAB built-in function `cholupdate.m`.

4.3 (*In class*).

1. In this exercise, we incorporate the Cholesky rank-one update procedure into the code developed in Exercise 1.3. Download and complete changeResistance_r1.m:

   ```
   r = changeResistance_r1(A,i,j,k);
   ```

   It has the same inputs and outputs as `changeResistance.m`, and it uses the Cholesky rank-one update in the loop instead of computing Cholesky factors afresh. You will need to figure out what the x is when an edge is added to the graph.

   Experiment with your code using $i = 500$, $j = 3000$ and $k = 10$. You may compare your output against changeResistance_r1_soln.p. How does it compare with the method in Exercise 1.3 in terms of speed? You may also try to use the MATLAB built-in function `cholupdate.m` in place of `mycholupdate.m`.

2. Another widely used method for obtaining $(B + xx^T)^{-1}$ given B^{-1} is the Sherman-Morrison formula. See also the Woodbury matrix identity for a generalization.

   Download and complete chol_vs_smw.m that computes $y = (L + ee^T + xx^T)^{-1}(e_i - e_j)$, where x corresponds to adding an edge between node i and a random neighbor of j:

   ```
   [errchol,errsmw] = chol_vs_smw(A,i,j);
   ```

   In this code, we assume knowledge of the Cholesky factor R of $L + ee^T$, and compare the additional cost needed to obtain y from performing the Cholesky update or using the Sherman-Morrison formula. The outputs `errchol` and `errsmw` are residuals

   $$\|(L + ee^T + xx^T)y - (e_i - e_j)\|,$$

   with the y obtained from the corresponding approach.

   You may compare your code against chol_vs_smw_soln.p. You may also try to use the MATLAB built-in function `cholupdate.m` in place of `mycholupdate.m`.

Five

Poisson Image Editing


Figure 5.1: Copy-and-paste (left) and smoothed (right) images.

1 Learning goals

In this case study, we discuss an unconventional application of the method of least squares: replacing part of an image by another while smoothing the transition across the seam. We expect that students will be familiar with eigenvalue decomposition for real symmetric matrices. The students will be guided step-by-step to solve the normal equation for the least-squares problem that arise. There are two twists that may be new for the students. First, the least-squares problem involves a matrix variable, and the linear maps in concern are defined by the left- and right-matrix multiplications. Thus, the students can see a least-squares problem in a different form than they might be used to. To minimize technicality, instead of having the students derive the normal equation, we simply point out that the normal equation is a special instance of a Lyapunov equation. Second, the students will be guided to use eigenvalue decomposition and properties of the Krönecker product to solve the Lyapunov equation. The Krönecker product is an important topic which typically is not discussed in introductory linear algebra courses due to time constraints.

2 Background

In image editing, one typical scenario would be to glue part of one image into another. One challenging task is to make the transition across the edge of the "plugged-in" image smooth. Compare the two pictures in figure 1 above. The one on the left is obtained by simply gluing the image of the cameraman onto the picture. The transition across the edges is very clear. The one on the right is obtained after applying the Poisson image editing technique to be discussed in this case study, and one can see much smoother transition across the edges.

3 Problem description

An image can be thought of as a function f defined on a box R (imagine you have infinitely refined pixels) with values in the range $[0, 255]$. Gluing two images amounts to replacing f in a particular region Ω by another function h. In Poisson image editing, we use gradients as guides: find a new function U whose gradients (in x and $-y$ directions) equal

$$G = (G^x, G^y) = \begin{cases} (\nabla_x h, -\nabla_y h) & \text{if } (x, y) \in \Omega, \\ (\nabla_x f, -\nabla_y f) & \text{otherwise.} \end{cases} \tag{5.1}$$

Since U is differentiable, intuitively, the transition across the boundary of Ω in the corresponding image should look smooth. However, there is no guarantee such a function would exist in general. Hence, we look for the best approximation in the least-squares sense:

$$\min_U \iint_R |(\nabla_x U, -\nabla_y U) - G|^2 dxdy.$$

For unique solvability, one has to impose a boundary condition. We leave out this technical detail for simplicity. Our solution is hence only determined up to an additive constant.

We focus on square images and discretize this integral for numerical computations:

$$\min_{U \in \mathbb{R}^{n \times n}} \|UD^T - G^x\|_F^2 + \|DU - G^y\|_F^2, \tag{5.2}$$

where D is the matrix representation of the (forward) finite difference operator, U is a matrix representing an image; and, with an abuse of notation, in (5.2), we use G^x and G^y to denote the discretized version of the gradients in (5.1). The actions of D, described in MATLAB notation, are

$$\begin{aligned} UD^T &= \big[U(:, i+1) - U(:, i)\big]_{i=1}^{n-1} \in \mathbb{R}^{n \times (n-1)}, \\ DU &= \big[U(i+1, :) - U(i, :)\big]_{i=1}^{n-1} \in \mathbb{R}^{(n-1) \times n}. \end{aligned} \tag{5.3}$$

4 Exercises

4.1 (*In class).*

1. Download the image files lake.tif and cameraman.tif (taken from the Image Database). Store the files in a directory called "data". Use the following commands to load and process the files. And don't forget the useful command `imshow(f,[])` for showing pictures.

```
f = imread('lake.tif');
h = imread('cameraman.tif');
f = double(f);
h = double(h);
```

2. Write a MATLAB function to construct D for any given size n.

```
D = constructD(n);
```

You may compare your output against `constructD_soln.p`.

Compute and display fD^T and Df. These will be the "discrete gradients" of f in the x and $-y$ directions, i.e., $fD^T \approx \nabla_x f$ and $Df \approx -\nabla_y f$.

3. Download and complete the code `mergeimage.m`.

```
[Gx,Gy] = mergeimage(f,h);
```

This code takes as input two images f and h of the same size. It allows the user to choose a particular *rectangular* region Ω in the image h to replace the corresponding part in f. The region Ω is specified by two diagonal vertices selected by the user via `ginput`. The code outputs the discretized version of (5.1).

You may want to compare your code against `mergeimage_soln.p`.

4. To get an idea of what this will lead to, download `solveimage_soln.p` and run

```
u = solveimage_soln(Gx,Gy);
imshow(u,[]);
```

Notice that the selected part of h is merged into the picture f.

You will be guided to write your own `solveimage.m` in Exercise 2.

4.2 (*Homework).* It is known that the normal equation of the least squares problem (5.2) is

$$UD^TD + D^TDU = G^xD + D^TG^y, \tag{5.4}$$

an instance of continuous Lyapunov equation. This can be shown using more advanced linear algebra techniques, which is out of the scope of this case study.

This part of the case study is devoted to developing a MATLAB code for solving (5.4).

1. Let $D^T D = Q\Lambda Q^T$ be an eigenvalue decomposition of $D^T D$. With $W = Q^T U Q$, show that (5.4) can be reduced to

$$W\Lambda + \Lambda W = Q^T(G^x D + D^T G^y)Q. \tag{5.5}$$

   What is the rank of $D^T D$?

2. By using the vec operation and properties of Kronecker product, show further that there exists a diagonal matrix $\mathfrak{D}$ of size $n^2 \times n^2$ so that (5.5) is equivalent to

$$\mathfrak{D}\,\mathrm{vec}(W) = \mathrm{vec}(B), \tag{5.6}$$

   where $B = Q^T(G^x D + D^T G^y)Q$.

   (Hint: Rewrite the left hand side of (5.5) as $IW\Lambda + \Lambda W I$ and use properties of Kronecker product.)

   What are the diagonal entries of $\mathfrak{D}$? How many of them are zero?

3. While there are infinitely many solutions to (5.6) due to the existence of zero diagonal entries in $\mathfrak{D}$, one solution can be obtained by using the pseudo-inverse of $\mathfrak{D}$.

   Download and complete the code `solveimage.m` which uses the above approach to solve (5.4). Your inputs are the G^x and G^y obtained in Exercise 1.

```
u = solveimage(Gx,Gy);
```

   (Hint: You should **not** form the diagonal matrix $\mathfrak{D}$ explicitly. You only need the diagonal entries!)

   You may want to compare your code against `solveimage_soln.p`. After that, you may also want to take a look at the MATLAB built-in function `lyap.m`.

4.3 (*In class*). In this exercise, we look at another way to merge/mix/modify the gradient G to produce interesting/useful output images. Download and complete the code `miximage.m`.

```
[Gx,Gy] = miximage(f,h);
```

The inputs are the same as those of `mergeimage.m`. Within the selected region Ω, the code modifies the discrete gradients as

$$G^x = \begin{cases} \nabla_x f & \text{if } |\nabla_x f| > |\nabla_x h|, \\ \nabla_x h & \text{otherwise,} \end{cases} \quad \text{and} \quad G^y = \begin{cases} -\nabla_y f & \text{if } |\nabla_y f| > |\nabla_y h|, \\ -\nabla_y h & \text{otherwise.} \end{cases}$$

You may want to compare your code against `miximage_soln.p`, and view the corresponding output from `solveimage.m`. What happened to the image(s)?

Compute the residual $\|UD^T - G^x\|_F^2 + \|DU - G^y\|_F^2$ and comment on the solvability of the system

$$\begin{aligned} UD^T &= G^x, \\ DU &= G^y. \end{aligned}$$

Six

Kalman Mouse Tracking

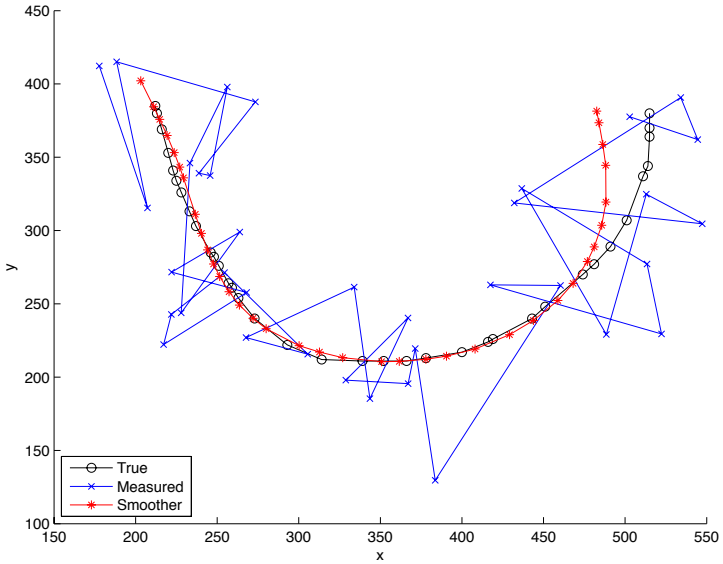


Figure 6.1: Using Kalman smoothing we can compute estimates of the true mouse cursor positions from noisy measurements.

1 Learning goals

In this case-study, we will learn how to use Kalman smoothing to estimate true mouse cursor positions from noisy measurements. Students will complete a graphical user interface written in MATLAB that obtains mouse positions, adds random noise to the positions, and computes the Kalman smoothed positions.

We formulate the Kalman smoothing problem as a generalized least-squares problem. We will learn how to solve this least-squares problem using Cholesky factorizations of the covariance matrices that model the correlated errors. Alternatively, the least-squares problem can be solved using the normal equations.

We assume students will already be familiar with solving linear least-squares in MATLAB and with the Cholesky factorization of symmetric positive definite matrices. In this case-study, students will learn how to work with block-matrices to build the generalized least-squares problem.

2 Background

We will see how we can use *Kalman smoothing* to estimate the position of a mouse cursor from noisy measurements. At times $t_1, t_2, \ldots, t_n$, we will make a noisy measurement of the position of the mouse cursor. We will assume that the *state* s_k of the mouse cursor at time t_k is given by its position $p_k = (x_k, y_k)$ and its velocity vector $v_k = (v_k^x, v_k^y)$. Thus, the state of the cursor at time t_k is given by the vector

$$s_k := \begin{bmatrix} p_k \\ v_k \end{bmatrix} \in \mathbb{R}^4, \qquad \text{for } k = 1, \ldots, n.$$

Moreover, we will assume that s_k can be approximated from the previous state s_{k-1} as

$$s_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} \approx \begin{bmatrix} p_{k-1} + \Delta t_{k-1} v_{k-1} \\ v_{k-1} \end{bmatrix} = A_k s_{k-1},$$

where

$$\Delta t_{k-1} := t_k - t_{k-1} \qquad \text{and} \qquad A_k := \begin{bmatrix} I & \Delta t_{k-1} I \\ 0 & I \end{bmatrix}, \qquad \text{for } k = 2, \ldots, n.$$

In addition, we assume we are given an estimate s_0 for the initial state s_1 and that $s_1 \approx s_0 = A_1 s_0$, where $A_1 := I$. Thus,

$$s_k \approx A_k s_{k-1}, \qquad \text{for } k = 1, \ldots, n.$$

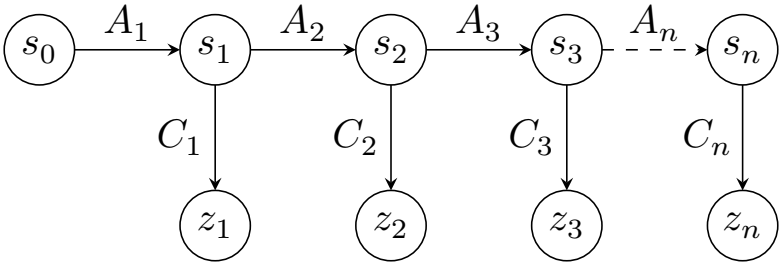
However, we will only be able to make a noisy *measurement* z_k of the position of the mouse cursor at each time t_k. That is,

$$z_k \approx p_k = C_k s_k,$$

where

$$C_k := \begin{bmatrix} I & 0 \end{bmatrix}, \qquad \text{for } k = 1, \ldots, n.$$

We can picture the entire problem with the following diagram.



Thus, given an estimate of the initial state s_0, we would like to compute estimates of the states $s_1, \ldots, s_n$ that best fit the measurements $z_1, \ldots, z_n$.

3 Problem description

We statistically model the problem as

$$\begin{cases} s_k = A_k s_{k-1} + \mu_k, & \mu_k \sim N(0, Q_k), \\ z_k = C_k s_k + \nu_k, & \nu_k \sim N(0, R_k), \end{cases}$$

where Q_k and R_k are positive definite matrices, for $k = 1, \ldots, n$. That is, μ_k and ν_k are vectors of random Gaussian variables with zero mean and known covariance matrices Q_k and R_k, respectively. Based on this statistical model, we want to solve the following *least-squares* optimization problem

$$\underset{s=(s_1,\ldots,s_n)}{\text{minimize}} \quad f(s) := \sum_{k=1}^{n} \|s_k - A_k s_{k-1}\|^2_{Q_k^{-1}} + \|z_k - C_k s_k\|^2_{R_k^{-1}}, \tag{6.1}$$

where the M-norm of a vector x is defined as $\|x\|_M := \sqrt{x^T M x}$.

For the estimate of the initial state and its corresponding covariance matrix, we will use

$$s_0 = \begin{bmatrix} p_0 \\ v_0 \end{bmatrix} := \begin{bmatrix} z_1 \\ 0 \end{bmatrix} \quad \text{and} \quad Q_1 := \text{diag}([\sigma_r^2, \sigma_r^2, 100, 100]),$$

for some $\sigma_r > 0$. The other covariance matrices are defined as

$$Q_k := \sigma_q^2 \begin{bmatrix} \frac{1}{3}\Delta t_{k-1}^3 I & \frac{1}{2}\Delta t_{k-1}^2 I \\ \frac{1}{2}\Delta t_{k-1}^2 I & \Delta t_{k-1} I \end{bmatrix}, \quad \text{for } k = 2, \ldots, n,$$

for some $\sigma_q > 0$, and

$$R_k := \sigma_r^2 I, \quad \text{for } k = 1, \ldots, n.$$

Thus, σ_r^2 is the variance associated with the *measurement error* and σ_q^2 is the variance associated with the *state-transition error*.

4 Exercises

4.1 (*In class).*

(a) We can re-write problem (6.1) as

$$\underset{s}{\text{minimize}} \quad f(s) = \|w - As\|^2_{Q^{-1}} + \|z - Cs\|^2_{R^{-1}}, \tag{6.2}$$

where s is a vector of length $4n$, w is a vector of length $4n$, A is $4n \times 4n$ matrix, Q is a $4n \times 4n$ positive definite matrix, z is a vector of length $2n$,

C is a $2n \times 4n$ matrix, and R is a $2n \times 2n$ positive definite matrix. Show that

$$A = \begin{bmatrix} I & & & \\ -A_2 & I & & \\ & \ddots & \ddots & \\ & & -A_n & I \end{bmatrix}, \quad s = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}, \quad w = \begin{bmatrix} s_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix},$$

and that

$$Q = \begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_n \end{bmatrix}, \quad R = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_n \end{bmatrix}, \quad C = \begin{bmatrix} C_1 & & \\ & \ddots & \\ & & C_n \end{bmatrix}.$$

(b) Let U and V be the Cholesky factors of the positive definite matrices Q and R, respectively. Show that

$$\|w - As\|^2_{Q^{-1}} = \|U^{-T}(w - As)\|_2^2 \quad \text{and} \quad \|z - Cs\|^2_{R^{-1}} = \|V^{-T}(z - Cs)\|_2^2,$$

then conclude that $\|w - As\|^2_{Q^{-1}} + \|z - Cs\|^2_{R^{-1}} = \|b - Ms\|_2^2$, where

$$b = \begin{bmatrix} U^{-T}w \\ V^{-T}z \end{bmatrix} \quad \text{and} \quad M = \begin{bmatrix} U^{-T}A \\ V^{-T}C \end{bmatrix}.$$

Therefore, problem (6.2) reduces to the least-squares problem

$$\underset{s}{\text{minimize}} \quad \|b - Ms\|_2^2. \tag{6.3}$$

(c) Write a MATLAB function

```
function s = myleastsquares(A, Q, C, R, w, z)
```

that computes a solution s to the optimization problem (6.2). Write a script `testmyleastsquares.m` that generates random matrices and tests your MATLAB function.

(d) Another way to solve the least-squares problem (6.2) is by solving the *normal equations*. Show that the optimality condition for the problem (6.2), $\nabla f(s) = 0$, corresponds to the following *normal equations*:

$$\left(A^T Q^{-1} A + C^T R^{-1} C\right) s = \left(A^T Q^{-1} w + C^T R^{-1} z\right).$$

4.2 (*Homework).* In MATLAB, load `data.mat`, compute the Kalman smoothed positions, and plot the results using the following commands:

```
load data
S = kalman_smoother(t,p,s0,Q1,sigq,sigr);

clf; hold on
plot(x,y,'ko-');           % plot true mouse positions
plot(p(:,1),p(:,2),'bx-'); % plot noisy measurements
plot(S(:,1),S(:,2),'r*-'); % plot smoothed positions
axis equal; axis([150 680 30 380]);
xlabel('x'); ylabel('y');
legend('True','Measured','Smoother',...
       'Location','SouthEast');
```

Currently the MATLAB function kalman_smoother.m simply returns the noisy measured positions with velocities set to zero.

```
function S = kalman_smoother(t, p, s0, Q1, sigq, sigr)
%KALMAN_SMOOTHER Smooths a 2D noisy signal.
%
% S = KALMAN_SMOOTHER(t, p, s0, Q1, sigq, sigr)
% smooths the 2D noisy signal given by (t,p) and
% returns the resulting signal S.
%
% t is the vector of measurement times.
% p is the n-by-2 matrix of 2D measurements.
% s0 is the estimate of the initial state.
% Q1 is the covariance matrix of the initial state
% estimate.
% sigq^2 is the variance associated with the state-
% transition error.
% sigr^2 is the variance associated with the
% measurement error.
```

Complete this function by modifying it to form and solve the least-squares problem (6.2) using the MATLAB function you wrote for Exercise 1. You may compare your output against the solution `kalman_smoother_soln.p`.

4.3 (*In class*). Download the MATLAB function `gui_mouse.m` and associated figure file `gui_mouse.fig` and test how it functions with the Kalman smoother you wrote in Exercise 2. Save different data sets and experiment with different levels of noise σ_r. How much noise is tolerated by the Kalman smoother? Experiment with different values of σ_q. What happens as σ_q becomes large?

Seven

Portfolio Optimization

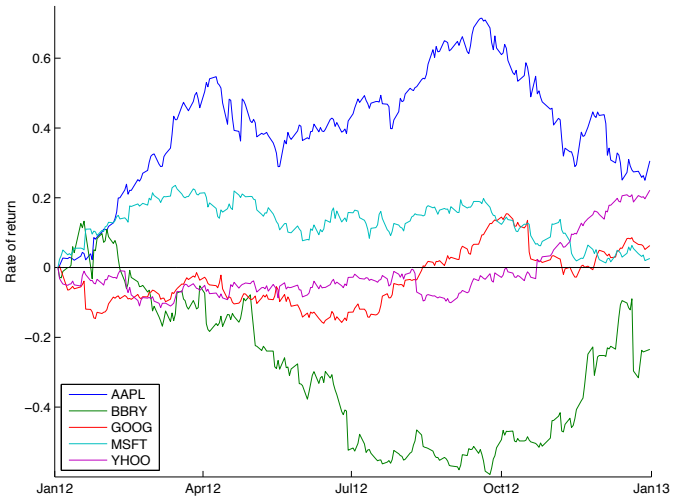


Figure 7.1: The historical rate of return of five technology stocks from the beginning of 2012.

1 Learning goals

In this case-study, we will learn how to compute portfolios of investments having either a maximum expected rate of return or a minimum overall volatility (i.e., risk) using real historical stock prices.

The exercises in this case-study will take us through an investigation of the relationship between the expected rate of return of a portfolio and its risk. We will start by downloading financial data from the Internet and using it to make a scatter plot of random portfolios. We will see that all portfolios lie in a region that is bounded by a curve called the *efficient frontier*. The portfolios on the efficient frontier are optimal in the sense that they maximize the expected rate of return for a fixed level of risk, and they minimize risk for a fixed level

of expected rate of return. Finally, we will make use of dual multipliers to compute the *market portfolio* associated with a zero-risk investment.

This case-study will introduce students to the important class of *convex optimization problems*. We will learn how we can use the MATLAB-based convex optimization modeling system CVX to model and solve portfolio optimization problems. We expect students to have a good working knowledge of MATLAB, but we do not assume that students have studied portfolio theory or convex optimization before.

2 Background

Modern portfolio theory is based on the Markowitz model for determining a portfolio of stocks with a desired expected rate of return that has the smallest amount of variance. The main idea is that by *diversifying* (investing in a mixture of different stocks), one can guard against large amounts of variance in the rates of return of the individual stocks.

Suppose $p_1, \ldots, p_m$ are the historical prices of a stock over some period of time. We define the *rate of return* at time t, relative to the initial price p_1 by

$$r_t := (p_t - p_1)/p_1, \quad \text{for} \quad t = 1, \ldots, m. \tag{7.1}$$

The *expected rate of return* is the mean μ of the rates of return, and the *risk* is defined as the *standard deviation* σ of the rates of return:

$$\mu := \frac{1}{m}\sum_{t=1}^{m} r_t \quad \text{and} \quad \sigma := \sqrt{\frac{1}{m}\sum_{t=1}^{m}(r_t - \mu)^2}.$$

Given a collection of n stocks, let r_t^i be the rate of return of stock i at time t. Let r be the $n \times 1$ vector of the expected rates of return of the n stocks. In addition, let Σ be the $n \times n$ *covariance matrix* of the rates of return of the n stocks. Thus, r_i is the mean of the rates of return of stock i, Σ_{ii} is the variance of the rates of return of stock i, and Σ_{ij} is the covariance of the rates of return of stocks i and j:

$$r_i := \frac{1}{m}\sum_{t=1}^{m} r_t^i \quad \text{and} \quad \Sigma_{ij} := \frac{1}{m}\sum_{t=1}^{m}(r_t^i - r_i)(r_t^j - r_j).$$

We let x_i be the fraction of our investment money we put into stock i, for $i = 1, \ldots, n$. For the sake of this study, we assume there is no *short selling* (i.e., holding a stock in negative quantity). Thus, x is a vector of length n that has nonnegative entries that sum to one (i.e., $x \geq 0$ and $\sum_{i=1}^{n} x_i = 1$). The vector x represents our *portfolio* of investments. The expected rate of return and standard deviation of a portfolio x are then given by

$$\mu := r^T x \quad \text{and} \quad \sigma := \sqrt{x^T \Sigma x}.$$

Note that Σ is positive semidefinite, so there exists an $n \times n$ matrix R such that $\Sigma = R^T R$; if Σ is positive definite, then we can choose R to be the Cholesky factor of Σ. Then we can compute the standard deviation of portfolio x as

$$\sigma = \|Rx\|_2.$$

3 Problem description

We will consider various optimization problems in the following exercises. First, we will be interested in finding the portfolio x that has the minimum possible risk:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sqrt{x^T \Sigma x} \\ \text{subject to} & \sum_{i=1}^{n} x_i = 1, \\ & x \geq 0. \end{array}$$

However, if we are seeking a minimum risk portfolio having expected return of at least μ, then we would want to solve:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sqrt{x^T \Sigma x} \\ \text{subject to} & r^T x \geq \mu, \\ & \sum_{i=1}^{n} x_i = 1, \\ & x \geq 0. \end{array}$$

On the other hand, we may have a maximum risk that we are willing to tolerate. In this case, we want to find the portfolio x having maximum expected return having risk no more than σ:

$$\begin{array}{ll} \underset{x}{\text{maximize}} & r^T x \\ \text{subject to} & \sqrt{x^T \Sigma x} \leq \sigma, \\ & \sum_{i=1}^{n} x_i = 1, \\ & x \geq 0. \end{array}$$

Note that each of these problems are convex optimization problems.

4 Exercises

Software prerequisites. You will need to install CVX, a MATLAB-based modeling system for convex optimization.

4.1 (*In class*).

1. Download financial data (`csv` files) from Yahoo! Finance for the following twenty stocks:

   - Technology: AAPL, BBRY, GOOG, MSFT, YHOO
   - Services: AMZN, COST, EBAY, TGT, WMT
   - Financial: BMO, BNS, HBC, RY, TD
   - Energy: BP, CVX, IMO, TOT, XOM

   Store the `csv` files in a directory called '`data`'.

2. Download and complete the function `load_stocks.m`:

   ```
   [X, dates, names] = load_stocks(dirname, startdate,
       enddate)
   ```

   This function must read the *adjusted closing prices* of all stocks in the given directory between the start date and end date, and compute the rates of return as in equation (7.1).

   Use the following start and end dates:

   ```
   startdate = '2012-01-03';  enddate = '2012-12-31';
   ```

   Plot your results using `disp_stocks.m`:

   ```
   disp_stocks(X, dates, names)
   ```

   You may compare your output against the solution `load_stocks_soln.p`.

3. Create a function `meancov.m` that returns the $n \times 1$ vector `r` of means and the $n \times n$ covariance matrix `Sig` of the rates of returns of n stocks given by `X`:

   ```
   [r, Sig] = meancov(X)
   ```

4. Download and complete the function `portfolio_scatter.m`:

   ```
   h = portfolio_scatter(r, Sig, num)
   ```

   This function must generate random portfolios and make a scatter plot of their expected rates of return and standard deviation. Each random portfolio is generated by randomly allocating a fraction of the overall investment among a small set of 5 randomly chosen stocks. Make a scatter plot with `num = 1000` points. This function returns a handle `h` to the figure. You may compare your output against the solution `portfolio_scatter_soln.p`.

4.2 (*Homework).*

1. Use CVX to compute the portfolio with minimum risk. What is the expected rate of return and standard deviation of this portfolio? Plot the rate of return of this portfolio over the entire time period. What is the portfolio with maximum possible expected rate of return? Create a function `return_range.m` that returns `num` linearly spaced rates of return between the rate of return of the portfolio with minimum risk and the maximum possible rate of return:

   ```
   rrange = return_range(r, Sig, num)
   ```

   You may compare your output against the solution `return_range_soln.p`.

2. Given a desired expected rate of return, we can see from the scatter plot that there are many portfolios that we can choose that have this expected rate of return. However, each of these portfolios have a different level of risk, or standard deviation. Among these, the most *efficient* portfolio is the one giving us the least amount of risk.

   Each expected rate of return determines a different efficient portfolio. Plotting the expected rate of return and standard deviation of each of the efficient portfolios will give us a curve called the *efficient frontier*.

   Download and complete the function `efficient_frontier.m`:

   ```
   [Y, rates, sigs] = efficient_frontier(r, Sig, num)
   ```

   This function will compute `num` efficient portfolios with linearly spaced rates of return (obtained from `return_range.m`). These portfolios will be stored in the `n`×`num` matrix `Y`, and their corresponding expected rates of return and standard deviation in vectors `rates` and `sigs`. Plot `sigs` and `rates` on the scatter plot, with `num = 12`:

   ```
   h = portfolio_scatter(r, Sig, 1000);
   [Y, rates, sigs] = efficient_frontier(r, Sig, num);
   figure(h); hold on; plot(sigs, rates, 'ro-');
   ylim([0 0.5]); xlim([0 max(sigs)]);
   ```

   Display your results using `disp_portfolios.m`:

   ```
   h = disp_portfolios(Y, rates, sigs, names)
   ```

   You may compare your output against the solution `efficient_frontier_soln.p`.

4.3 (*In class).* Add a risk-free investment called 'RF' to the collection of stocks with a 3% rate of return. Use your `efficient_frontier.m` code from Exercise 2 to determine the new efficient frontier and plot it on the same plot with the original efficient frontier. You will notice that the new efficient frontier

has two pieces: (1) a linear piece, and (2) a nonlinear piece that coincides with the original efficient frontier. What does the linear piece represent? The portfolio where these two pieces join is called the *market portfolio*. Download and complete the function `market_portfolio.m` that computes the market portfolio corresponding to a risk-free rate of return `f`:

```
x = market_portfolio(f, r, Sig)
```

You may compare your output against the solution `market_portfolio_soln.p`. Plot the line that is tangent to the original efficient frontier at the market portfolio. What does the top half of this tangent line represent?

Eight

Image Deblurring

Figure 8.1: Original (left) and blurred (right) images.

1 Learning goals

This case study outlines an image deblurring problem. The blurring process is modeled by an action of a blurring matrix, and the deblurring/recovery process involves taking the "inverse" of the blurring matrix. We expect that students will be familiar with singular value decompositions of a matrix. In this case study, the students will be guided through to see that the blurring matrix is invertible but the smallest singular value is nearly zero. Consequently, taking inverse is an infeasible deblurring approach. We discuss two different remedies, which should be new for the students. The first one is the Tikonov method. The students will be required to derive a formula for the image recovered via the Tikonov method and observe how the procedure helps in suppressing measurement noise. The second method is to truncate small singular values and use the pseudo-inverse of the modified matrix. The students will also be asked to implement this latter method by themselves to see how deblurring works.

2 Background

Images captured by cameras are often blurry; for example, the lens may not be focused, or the cameraman's hand could be shaky. One important problem in image processing is thus to *deblur* an image. A simple idea is to model the blurred image as an output of a linear map (i.e., multiplication by a matrix) on the original image. Under that assumption, the process of deblurring essentially involves solving a system of linear equations.

In this case study, under a simplifying assumption that the blurring process is also separable, we will look at how the deblurring process is affected by the measurement noise in the images, and discuss two remedies based on modifying the small singular values of the blurring matrix.

3 Problem description

For simplicity, we only consider greyscale images. Such an image can be regarded as a two-dimensional array, where each entry corresponds to the *intensity* of a pixel in the image. Typical values for intensity are integers from 0 to 255, with 0 meaning black and 255 meaning white. Let $\boldsymbol{X}_{\text{true}} \in \mathbb{R}^{m\times n}$ denote the true image and $\boldsymbol{Y} \in \mathbb{R}^{m\times n}$ denote the blurred image. We assume the blurring process is both *linear* and *separable*. Specifically,

$$\boldsymbol{Y} = A_c \boldsymbol{X}_{\text{true}} A_r^T + \epsilon, \tag{8.1}$$

for some *blurring matrices* $A_c \in \mathbb{R}^{m\times m}$ and $A_r \in \mathbb{R}^{n\times n}$, and a matrix ϵ that captures measurement errors; we will assume that each entry of the matrix is normally distributed, i.e., $\epsilon_{ij} \sim N(0, \sigma^2)$ for some $\sigma > 0$. This is a special case of the general model

$$\operatorname{vec} \boldsymbol{Y} = \mathcal{A} \operatorname{vec} \boldsymbol{X}_{\text{true}} + \tilde{\epsilon}, \tag{8.2}$$

where vec is the operation that reshapes a matrix into a vector by stacking the columns, $\mathcal{A} \in \mathbb{R}^{mn\times mn}$ and $\tilde{\epsilon}$ is a vector. Relation (8.1) can be seen as a special case of (8.2) by taking $\tilde{\epsilon} = \operatorname{vec} \epsilon$ and $\mathcal{A} = A_r \otimes A_c$, where $\otimes$ denotes the Kronecker product.

To understand the action of the blurring matrices, imagine you are taking a picture of a point source, i.e., a picture $\boldsymbol{X}_0$ that is nonzero only at entry (i, j) and zero everywhere else. It is not hard to see that $A_c \boldsymbol{X}_0$ can only have nonzero entries along the jth column. Thus, the matrix A_c describes how blurring occurs in the vertical direction. Similarly, A_r describes how blurring occurs in the horizontal direction.

Given the blurring matrices A_c and A_r, one naive approach to recover $\boldsymbol{X}_{\text{true}}$ given the blurred image Y is to apply the "inverse" of A_c and A_r^T to both sides of (8.1). Unfortunately, as we will see below, even when ϵ is moderately small, the noise term tends to dominate in the deblurred image.

4 Exercises

4.1 (*In class).*

1. Download the image file `jetplane.tif` (taken from the Image Database), and the blurring matrices (`mat` file). Store the files in a directory called "data". Use the following commands to load, process and display the image file.

```
G = imread('jetplane.tif');
G = double(G);
imshow(G,[])
```

2. Write a function `Blur.m`

```
Gblur = Blur(G, Ac, Ar, sigma)
```

This function outputs the blurred image according to (8.1). It takes as input the image G, the blurring matrices A_c and A_r, and the noise level σ.

Obtain images G_σ, for $\sigma = 0$ and 0.05. You may use `imshow` to display them.

3. What is the formula for the naive approach for deblurring described in the introduction? Write a function `Deblur.m` that deblurs according to this approach.

```
Gdeblur = Deblur(Gblur, Ac, Ar)
```

This function takes in the blurred image G_σ and the blurring matrices A_c and A_r. You may compare your output against the solution `Deblur_soln.p`. Try this function on G_0 and $G_{0.05}$. How does it work?

Plot the root mean square error between the deblurred image and the true image against $\sigma = 0, 0.005, \ldots, 0.05$. Compare the errors with the magnitude of the entries of G_σ.

4.2 (*Homework).*

1. In this exercise, you will construct blurring matrices A_c and A_r. The matrix A_c, for example, describes blurring along the vertical direction, and its kth column can be extracted by multiplication with the vector e_k, which has a single 1 at the kth entry, and is zero otherwise. The kth column of this matrix describes the blurring effect on a "light source" at the kth position. Gaussian blurring redistributes the single light source according to a Gaussian distribution. Hence,

$$(A_c e_k)_i = f_\tau(i-k) := \frac{1}{\sqrt{2\pi\tau}} \exp\left(-\frac{(i-k)^2}{2\tau}\right), \tag{8.3}$$

where $\tau > 0$ is the variance. Make a plot of $f_{2.5}(t)$ for $t \in [-10, 10]$ and comment on how the function value decays.

Create a function

```
A = Makeblurmatrix(n, tau)
```

that generates an $n \times n$ blurring matrix as described above. A_r is generated in exactly the same way. You can compare your output against `Makeblurmatrix_soln.p`.

The blurring matrix generated this way is said to satisfy the zero-boundary condition because spread of light from light sources outside the image is ignored.

2. In this exercise, we discuss the Tikonov method, one way to refine our naive deblurring strategy. In this approach, instead of solving (8.2), we solve

$$\min_x \quad \tfrac{1}{2}\|\mathcal{A}x - y\|^2 + \tfrac{\alpha}{2}\|x\|^2,$$

where $y = \mathcal{A}\,\mathrm{vec}\,\boldsymbol{X}_{\mathrm{true}} + \tilde{\epsilon}$ and the variable x corresponds to $\mathrm{vec}\,\boldsymbol{X}$. It can be shown that solving the above optimization problem is the same as solving

$$(\mathcal{A}^T\mathcal{A} + \alpha I)x = \mathcal{A}^T y.$$

When $\alpha = 0$, the above approach reduces to solving a linear least-square problem and the corresponding normal equation.

Let $\mathcal{A} = USV^T$ denote the singular value decomposition of the $n^2 \times n^2$ matrix $\mathcal{A}$. Show that x can be represented as

$$x = \left(\sum_{i=1}^{n^2} \frac{s_i^2}{s_i^2 + \alpha} v_i v_i^T\right) \mathrm{vec}\,\boldsymbol{X}_{\mathrm{true}} + \sum_{i=1}^{n^2} \frac{s_i u_i^T \tilde{\epsilon}}{s_i^2 + \alpha} v_i, \tag{8.4}$$

where u_i and v_i are the ith columns of U and V respectively, and s_i is the diagonal of S. What does the formula look like when $\alpha = 0$?

Notice that in the presence of noise and small singular values, when $\alpha = 0$, the 1st term on the right of (8.4) gives the true image, but the 2nd term would become dominant due to division by s_i. This was observed empirically in Exercise 1.3, for the separable blurring model (8.1). Setting $\alpha > 0$ prevents the 2nd term from becoming too large.

Create A_c and A_r using `Makeblurmatrix` with $\tau = 2.5$. Download the MATLAB code `Deblur_Tik_soln.p` which deblurs using the Tikonov method, for the model (8.1).

```
Gdeblur = Deblur_Tik_soln(Gblur, Ac, Ar, alpha)
```

On the same graph, for each α equal to `1e-6, 5e-6`, and `1e-5`, plot the root mean square error between the deblurred image and the true image against $\sigma = 0, 0.005, \ldots, 0.05$. Compare the errors with the magnitude of the entries of G_σ, as well as what you obtained in Exercise 1.

4.3 (*In class).*

1. Let $A_c = U_c S_c V_c^T$ and $A_r = U_r S_r V_r^T$ denote the singular value decompositions of A_c and A_r generated in the previous problem, where U_c, U_r, V_c and V_r are orthogonal matrices and S_c and S_r are diagonal matrices.

   A simpler way to deblur, which avoids division by small singular values in a way different from (8.4), is to zero out the small entries of S_c and S_r, and use the pseudo-inverses of the truncated matrices in place of inverses in the naive recovery approach.

   Download and complete `Deblur_tol.m` that deblurs this way. In this function, the entries in S_c (respectively, S_r) below `tol` are set to zero.

   ```
   Gdeblur = Deblur_tol(Gblur, Ac, Ar, tol)
   ```

   You may compare your output against the solution `Deblur_tol_soln.p`.

   On the same graph, for each `tol`= 0.01, 0.05 and 0.1, plot the root mean square error between the deblurred image and the true image against $\sigma = 0, 0.005, \ldots, 0.05$. Compare the errors with the magnitude of the entries of G_σ. How is the performance compared with `Deblur_Tik.m`?

2. Download the matrix mat file which records a blurred image. Use `imshow` as in Exercise 1 to view the picture. It is known that it is blurred by (8.1) via Gaussian blurring matrices (8.3), for some $A_c = A_r$, with small noise. Deblur the picture (with possibly a new set of blurring matrices) and identify what is on the license plate.