

Homework 5

Due Thursday 10/29

1. *Projection matrices.*

A matrix $P \in \mathbb{R}^{n \times n}$ is called a *projection matrix* if $P = P^T$ and $P^2 = P$.

- Show that if P is a projection matrix then so is $I - P$.
- Suppose that the columns of $U \in \mathbb{R}^{n \times k}$ are orthonormal. Show that UU^T is a projection matrix. (Later we will show that the converse is true: every projection matrix can be expressed as UU^T for some U with orthonormal columns.)
- Suppose $A \in \mathbb{R}^{n \times k}$ is full rank, with $k \leq n$. Show that $A(A^T A)^{-1}A^T$ is a projection matrix.
- If $S \subseteq \mathbb{R}^n$ and $x \in \mathbb{R}^n$, the point y in S closest to x is called the *projection of x on S* . Show that if P is a projection matrix, then $y = Px$ is the projection of x on $\text{range}(P)$. (Which is why such matrices are called projection matrices . . .)

2. *Gradient of some common functions.*

Recall that the gradient of a differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, at a point $x \in \mathbb{R}^n$, is defined as the vector

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

where the partial derivatives are evaluated at the point x . The first order Taylor approximation of f , near x , is given by

$$\hat{f}_{\text{tay}}(z) = f(x) + \nabla f(x)^T (z - x).$$

This function is affine, *i.e.*, a linear function plus a constant. For z near x , the Taylor approximation \hat{f}_{tay} is very near f . Find the gradient of the following functions. Express the gradients using matrix notation.

- $f(x) = a^T x + b$, where $a \in \mathbb{R}^n$, $b \in \mathbb{R}$.
- $f(x) = x^T A x$, for $A \in \mathbb{R}^{n \times n}$.
- $f(x) = x^T A x$, where $A = A^T \in \mathbb{R}^{n \times n}$. (Yes, this is a special case of the previous one.)

3. *Least-squares deconvolution.*

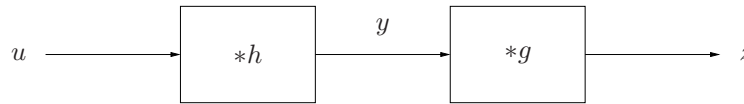
A communications channel is modeled by a finite-impulse-response (FIR) filter:

$$y(t) = \sum_{\tau=0}^{n-1} u(t - \tau)h(\tau),$$

where $u : \mathbb{Z} \rightarrow \mathbb{R}$ is the channel input sequence, $y : \mathbb{Z} \rightarrow \mathbb{R}$ is the channel output, and $h(0), \dots, h(n-1)$ is the impulse response of the channel. In terms of discrete-time convolution we write this as $y = h * u$. You will design a *deconvolution filter* or *equalizer* which also has FIR form:

$$z(t) = \sum_{\tau=0}^{m-1} y(t - \tau)g(\tau),$$

where $z : \mathbb{Z} \rightarrow \mathbb{R}$ is the filter output, y is the channel output, and $g(0), \dots, g(m-1)$ is the impulse response of the filter, which we are to design. This is shown in the block diagram below.



The goal is to choose $g = (g(0), \dots, g(m-1))$ so that the filter output is approximately the channel input delayed by D samples, *i.e.*, $z(t) \approx u(t-D)$. Since $z = g * h * u$ (discrete-time convolution), this means that we'd like

$$(g * h)(t) \approx \begin{cases} 0 & t \neq D, \\ 1 & t = D \end{cases}$$

We will refer to $g * h$ as the *equalized impulse response*; the goal is to make it as close as possible to a D -sample delay. Specifically, we want the *least-squares* equalizer is g that minimizes the sum-of-squares error

$$\sum_{t \neq D} (g * h)(t)^2,$$

subject to the constraint

$$(g * h)(D) = 1.$$

To solve the problem below you'll need to get the file `deconv_data.m` from the class web page in the *Matlab files* section. It will define the channel impulse response h as a Matlab vector `h`. (Indices in Matlab run from 1 to n , while the argument of the channel impulse response runs from $t = 0$ to $t = n - 1$, so `h(3)` in Matlab corresponds to $h(2)$.)

- Find the least-squares equalizer g , of length $m = 20$, with delay $D = 12$. Plot the impulse responses of the channel (h) and the equalizer (g). Plot the equalized impulse response ($g * h$).
- The vector y (also defined in `deconv_data.m`) contains the channel output corresponding to a signal u passed through the channel (*i.e.*, $y = h * u$). The signal u is binary, *i.e.*, $u(t) \in \{-1, 1\}$, and starts at $t = 0$ (*i.e.*, $u(t) = 0$ for $t < 0$). Pass y through the least-squares equalizer found in part a, to form the signal z . Give a histogram plot of the amplitude distribution of both y and z . (You can remove the first and last D samples of z before making the histogram plot.) Comment on what you find.

Matlab hints: The command `conv` convolves two vectors; the command `hist` plots a histogram (of the amplitude distribution).

4. Estimation with sensor offset and drift.

We consider the usual estimation setup:

$$y_i = a_i^T x + v_i, \quad i = 1, \dots, m,$$

where

- y_i is the i th (scalar) measurement
- $x \in \mathbb{R}^n$ is the vector of parameters we wish to estimate from the measurements
- v_i is the sensor or measurement error of the i th measurement

In this problem we assume the measurements y_i are taken at times evenly spaced, T seconds apart, starting at time $t = T$. Thus, y_i , the i th measurement, is taken at time $t = iT$. (This isn't really material; it just makes the interpretation simpler.) You can assume that $m \geq n$ and the measurement matrix

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}$$

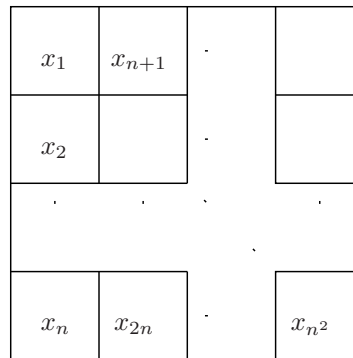
is full rank (*i.e.*, has rank n). Usually we assume (often implicitly) that the measurement errors v_i are random, unpredictable, small, and centered around zero. (You don't need to worry about how to make this idea precise.) In such cases, least-squares estimation of x works well. In some cases, however, the measurement error includes some *predictable* terms. For example, each sensor measurement might include a (common) *offset* or *bias*, as well as a term that grows linearly with time (called a *drift*). We model this situation as

$$v_i = \alpha + \beta iT + w_i$$

where α is the sensor bias (which is unknown but the *same* for all sensor measurements), β is the drift term (again the same for all measurements), and w_i is part of the sensor error that is unpredictable, small, and centered around 0. If we knew the offset α and the drift term β we could just subtract the predictable part of the sensor signal, *i.e.*, $\alpha + \beta iT$ from the sensor signal. But we're interested in the case where we don't know the offset α or the drift coefficient β . Show how to use least-squares to *simultaneously* estimate the parameter vector $x \in \mathbb{R}^n$, the offset $\alpha \in \mathbb{R}$, and the drift coefficient $\beta \in \mathbb{R}$. Clearly explain your method. If your method always works, say so. Otherwise describe the conditions (on the matrix A) that must hold for your method to work, and give a simple example where the conditions don't hold.

5. Image reconstruction from line integrals.

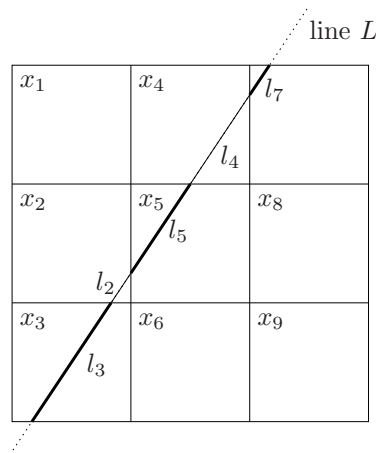
In this problem we explore a simple version of a tomography problem. We consider a square region, which we divide into an $n \times n$ array of square pixels, as shown below.



The pixels are indexed column first, by a single index i ranging from 1 to n^2 , as shown above. We are interested in some physical property such as density (say) which varies over the region. To simplify things, we'll assume that the density is constant inside each pixel, and we denote by x_i the density in pixel i , $i = 1, \dots, n^2$. Thus, $x \in \mathbb{R}^{n^2}$ is a vector that describes the density across the rectangular array of pixels. The problem is to estimate the vector of densities x , from a set of sensor measurements that we now describe. Each sensor measurement is a *line integral* of the density over a line L . In addition, each measurement is corrupted by a (small) noise term. In other words, the sensor measurement for line L is given by

$$\sum_{i=1}^{n^2} l_i x_i + v,$$

where l_i is the length of the intersection of line L with pixel i (or zero if they don't intersect), and v is a (small) measurement noise. This is illustrated below for a problem with $n = 3$. In this example, we have $l_1 = l_6 = l_8 = l_9 = 0$.



Now suppose we have N line integral measurements, associated with lines L_1, \dots, L_N . From these measurements, we want to estimate the vector of densities x . The lines are characterized by the intersection lengths

$$l_{ij}, \quad i = 1, \dots, n^2, \quad j = 1, \dots, N,$$

where l_{ij} gives the length of the intersection of line L_j with pixel i . Then, the whole set of measurements forms a vector $y \in \mathbb{R}^N$ whose elements are given by

$$y_j = \sum_{i=1}^{n^2} l_{ij} x_i + v_j, \quad j = 1, \dots, N.$$

And now the problem: you will reconstruct the pixel densities x from the line integral measurements y . The class webpage contains the M-file `tomodata.m`, which you should download and run in Matlab. It creates the following variables:

- `N`, the number of measurements (N),
- `n_pixels`, the side length in pixels of the square region (n),
- `y`, a vector with the line integrals y_j , $j = 1, \dots, N$,
- `lines_d`, a vector containing the displacement d_j , $j = 1, \dots, N$, (distance from the center of the region in pixels lengths) of each line, and
- `lines_theta`, a vector containing the angles θ_j , $j = 1, \dots, N$, of each line.

The file `tmeasure.m`, on the same webpage, shows how the measurements were computed, in case you're curious. You should take a look, but you don't need to understand it to solve the problem. We also provide the function `line_pixel_length.m` on the webpage, which you do need to use in order to solve the problem. This function computes the pixel intersection lengths for a given line. That is, given d_j and θ_j (and the side length n), `line_pixel_length.m` returns a $n \times n$ matrix, whose i, j th element corresponds to the intersection length for pixel i, j on the image. Use this information to find x , and display it as an image (of n by n pixels). You'll know you have it right when the image of x forms a familiar acronym... *Matlab hints:* Here are a few functions that you'll find useful to display an image:

- `A=reshape(v,n,m)`, converts the vector v (which must have $n*m$ elements) into an $n \times m$ matrix (the first column of A is the first n elements of v , etc.),
- `imagesc(A)`, displays the matrix A as an image, scaled so that its lowest value is black and its highest value is white,
- `colormap gray`, changes Matlab's image display mode to grayscale (you'll want to do this to view the pixel patch),

- `axis image`, redefines the axes of a plot to make the pixels square.

Note: While irrelevant to your solution, this is actually a simple version of *tomography*, best known for its application in medical imaging as the CAT scan. If an *x-ray* gets attenuated at rate x_i in pixel i (a little piece of a cross-section of your body), the j -th measurement is

$$z_j = \prod_{i=1}^{n^2} e^{-x_i l_{ij}},$$

with the l_{ij} as before. Now define $y_j = -\log z_j$, and we get

$$y_j = \sum_{i=1}^{n^2} x_i l_{ij}.$$