MCEN 4125/5125

Homework #2

Due: at the beginning of class on the following Thursday.

1) Show that the least-squares solution of $z^\star = (A^TA)^{-1}A^Td$, where

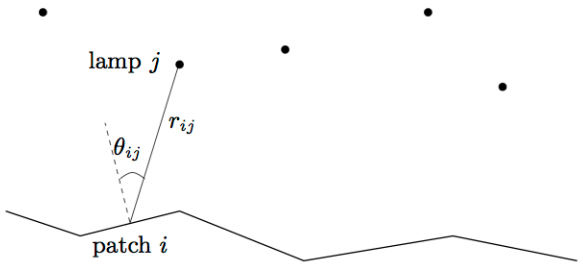
$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}; d = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}; z = \begin{bmatrix} m \\ b \end{bmatrix}$$

, results in the same solution as following:

$$m^\star = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}; b^\star = \frac{\overline{x^2}\bar{y} - \overline{xy}\bar{x}}{\overline{x^2} - \bar{x}^2}$$

Note that the bar over a vector ($\overline{xy}$) is the symbol for average (i.e. $\overline{xy} = \frac{1}{n}\sum_{i=1}^{n} x_i y_i$).

2) The figure shows an illumination system of n lamps illuminating m flat patches. The variables in the problem are the lamp powers $x_1, \ldots, x_n$, which can vary between 0 and 1.



The illumination intensity at (the midpoint of) patch i is denoted I_i. We will use a simple linear model for the illumination intensities I_i as a function of the lamp powers x_j: for $i = 1, \ldots, m$,

$$I_i = \sum_{j=1}^{n} a_{ij} x_j.$$

The matrix A (with coefficients a_{ij}) is available from the class webpage (see below), and was constructed as follows. We take

$$a_{ij} = r_{ij}^{-2} \max\{\cos\theta_{ij}, 0\},$$

where r_{ij} denotes the distance between lamp j and the midpoint of patch i, and θ_{ij} denotes the angle between the upward normal of patch i and the vector from the midpoint of patch i to lamp j, as shown in the figure. This model takes into account "self-shading" (*i.e.*, the fact that a patch is illuminated only by lamps in the halfspace it faces) but not shading of one patch caused by another. Of course we could use a more complex illumination model, including shading and even reflections. This just changes the matrix relating the lamp powers to the patch illumination levels.

The problem is to determine lamp powers that make the illumination levels I_i close to a given desired level $I_{\rm des}$. In other words, we want to choose the n-vector x such that

$$\sum_{j=1}^{n} a_{ij}x_j \approx I_{\rm des}, \quad i = 1, \ldots, m,$$

but we also have to observe the power limits $0 \leq x_j \leq 1$. This is an example of a *constrained optimization problem*. The objective is to achieve an illumination level that is as uniform as possible; the constraint is that the components of x must satisfy $0 \leq x_j \leq 1$. Finding the exact solution of this minimization problem requires specialized numerical techniques for constrained optimization. However, we can solve it *approximately* using least-squares.

In this problem we consider two approximate methods that are based on least-squares, and compare them for the data generated using `[A, Ides] = HW1Prob2`, with the MATLAB `HW1Prob2.m`. The elements of A are the coefficients a_{ij}. In this example we have $m = 11$, $n = 7$ so A is 11×7, and $I_{\rm des} = 2$.

(a) *Saturate the least-squares solution.* The first method is simply to ignore the bounds on the lamp powers. We solve the least-squares problem

$$\text{minimize} \quad \sum_{i=1}^{m}(\sum_{j=1}^{n} a_{ij}x_j - I_{\rm des})^2$$

ignoring the constraints $0 \leq x_j \leq 1$. If we are lucky, the solution will satisfy the bounds $0 \leq x_j \leq 1$, for $j = 1, \ldots, n$. If not, we replace x_j with zero if $x_j < 0$ and with one if $x_j > 1$.

Apply this method to the problem data generated by `ch8ex8.m`, and calculate the resulting value of the cost function $\sum_{i=1}^{m}(I_i - I_{\rm des})^2$.

(b) *Weighted least-squares.* The second method is to solve the problem

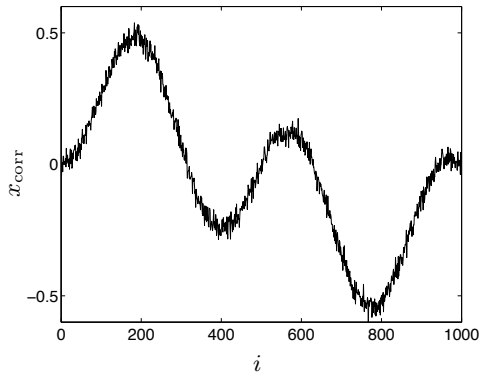
$$\text{minimize} \quad \sum_{i=1}^{m}(\sum_{j=1}^{n} a_{ij}x_j - I_{\rm des})^2 + \mu\sum_{j=1}^{n}(x_j - 0.5)^2,$$

where the constant $\mu \geq 0$ is used to attach a cost to the deviation of the powers from the value 0.5, which lies in the middle of the power limits. For $\mu = 0$, this is the same least-squares problem as in part (a). If we take μ large enough, the solution of this problem will satisfy $0 \leq x_j \leq 1$.

Formulate this problem as a least-squares problem in the variables x, and solve it for $\mu = 1$, $\mu = 2$, $\mu = 3$, etc., until you find a value of μ such that all components of the solution x satisfy $0 \leq x_j \leq 1$. For that solution x, calculate the cost function $\sum_{i=1}^{m}(I_i - I_{\rm des})^2$ and compare with the value you obtained in part (a).

3)

De-noising using least-squares. The figure shows a signal of length 1000, corrupted with noise. We are asked to estimate the original signal. This is called signal reconstruction, or de-noising, or smoothing. In this problem we apply a smoothing method based on least-squares.



We will represent the corrupted signal as a vector $x_{\rm cor}$ of size 1000. (The values can be obtained as `xcor = HW1Prob3` using the file `HW1Prob3.`) The estimated signal (*i.e.*, the variable in the problem) will be represented as a vector $\hat{x}$ of size 1000.

The idea of the method is as follows. We assume that the noise in the signal is the small and rapidly varying component. To reconstruct the signal, we decompose $x_{\rm cor}$ in two parts

$$x_{\rm cor} = \hat{x} + v$$

where v is small and rapidly varying, and $\hat{x}$ is close to $x_{\rm cor}$ ($\hat{x} \approx x_{\rm cor}$) and slowly varying ($\hat{x}_{i+1} \approx \hat{x}_i$). We can achieve such a decomposition by choosing $\hat{x}$ as the solution of the least-squares problem

$$\text{minimize} \quad \|x - x_{\rm cor}\|^2 + \mu \textstyle\sum_{i=1}^{999} (x_{i+1} - x_i)^2, \tag{8.10}$$

where μ is a positive constant. The first term $\|x - x_{\rm cor}\|^2$ measures how much x deviates from $x_{\rm cor}$. The second term, $\sum_{i=1}^{999}(x_{i+1} - x_i)^2$, penalizes rapid changes of the signal between two samples. By minimizing a weighted sum of both terms, we obtain an estimate $\hat{x}$ that is close to $x_{\rm cor}$ (*i.e.*, has a small value of $\|\hat{x} - x_{\rm cor}\|^2$) and varies slowly (*i.e.*, has a small value of $\sum_{i=1}^{999}(\hat{x}_{i+1} - \hat{x}_i)^2$). The parameter μ is used to adjust the relative weight of both terms.

Problem (8.10) is a least-squares problem, because it can be expressed as

$$\text{minimize} \quad \|Ax - b\|^2$$

where

$$A = \begin{bmatrix} I \\ \sqrt{\mu}\, D \end{bmatrix}, \qquad b = \begin{bmatrix} x_{\rm cor} \\ 0 \end{bmatrix},$$

and D is a 999×1000-matrix defined as

$$D = \begin{bmatrix} -1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 1 \end{bmatrix}.$$

The matrix A is quite large (1999×1000), but also very sparse, so we will solve the least-squares problem using the Cholesky factorization method. You should verify that the normal equations are given by

$$(I + \mu D^T D)x = x_{\text{cor}}. \tag{8.11}$$

MATLAB and Octave provide special routines for solving sparse linear equations, and they are used as follows. There are two types of matrices: full (or dense) and sparse. If you define a matrix, it is considered full by default, unless you specify that it is sparse. You can convert a full matrix to sparse format using the command `A = sparse(A)`, and a sparse matrix to full format using the command `A = full(A)`.

When you type `x = A\b` where A is $n \times n$, MATLAB chooses different algorithms depending on the type of A. If A is full it uses the standard method for general matrices (LU or Cholesky factorization, depending on whether A is symmetric positive definite or not). If A is sparse, it uses an LU or Cholesky factorization algorithm that takes advantage of sparsity. In our application, the matrix $I + \mu D^T D$ is sparse (in fact tridiagonal), so if we make sure to define it as a sparse matrix, the normal equations will be solved much more quickly than if we ignore the sparsity.

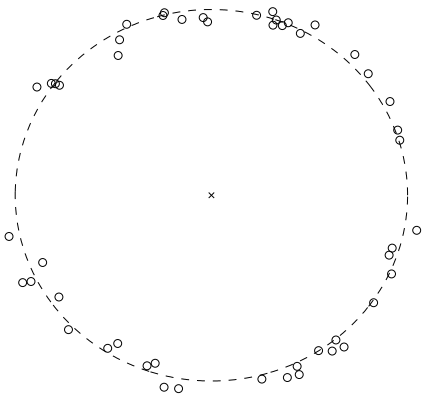
The command to create a sparse zero matrix of dimension $m \times n$ is `A = sparse(m,n)`. The command `A = speye(n)` creates a sparse $n \times n$-identity matrix. If you add or multiply sparse matrices, the result is automatically considered sparse.

This means you can solve the normal equations (8.11) by the following MATLAB code (assuming μ and x_{cor} are defined):

```
D = sparse(999,1000);
D(:,1:999) =  -speye(999);
D(:,2:1000) =  D(:,2:1000) + speye(999);
xhat = (speye(1000) + mu*D'*D) \ xcor;
```

Solve the least-squares problem (8.10) with the vector x_{cor} defined in `ch8ex9.m`, for three values of μ: $\mu = 1$, $\mu = 100$, and $\mu = 10000$. Plot the three reconstructed signals $\hat{x}$. Discuss the effect of μ on the quality of the estimate $\hat{x}$.

4) In this problem we use least-squares to fit a circle to given points (u_i, v_i) in a plane, as shown in the figure.



We use (u_c, v_c) to denote the center of the circle and R for its radius. A point (u, v) is on the circle if $(u - u_c)^2 + (v - v_c)^2 = R^2$. We can therefore formulate the fitting problem as

$$\text{minimize} \quad \sum_{i=1}^{m} \left((u_i - u_c)^2 + (v_i - v_c)^2 - R^2 \right)^2$$

with variables u_c, v_c, R.
Show that this can be written as a linear least-squares problem if we make a change of variables and use as variables u_c, v_c, and $w = u_c^2 + v_c^2 - R^2$.

(a) Define A, b, and x in the equivalent linear least-squares formulation.

(b) Show that the optimal solution u_c, v_c, w of the least-squares problem satisfies $u_c^2 + v_c^2 - w \geq 0$. (This is necessary to compute $R = \sqrt{u_c^2 + v_c^2 - w}$ from the result u_c, v_c, w.)

Test your formulation on the problem data in the file HW1Prob4.m
The commands

```
[u,v] = HW1Prob4
plot(u, v, 'o');
axis square
```

will create a plot of the $m = 50$ points (u_i, v_i) in the figure. The following code plots the 50 points and the computed circle.

```
t = linspace(0, 2*pi, 1000);
plot(u, v, 'o',  R * cos(t) + uc, R * sin(t) + vc, '-');
axis square
```

(assuming your MATLAB variables are called `uc`, `vc`, and `R`).