# Homework 4

EE263 Stanford University, Fall 2017

Due: Friday 10/27/17 11:59pm

1. **Estimation with sensor offset and drift.** We consider the usual estimation setup:

$$y_i = a_i^\mathsf{T} x + v_i, \qquad i = 1, \ldots, m,$$

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

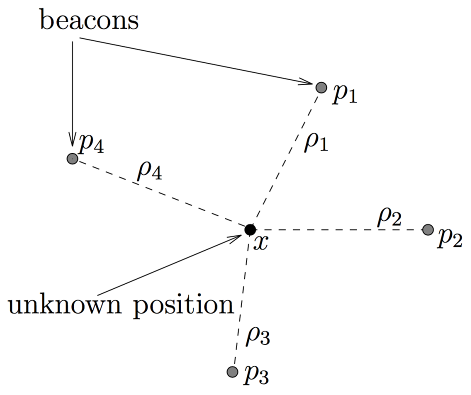
$$A = \begin{bmatrix} a_1^\mathsf{T} \\ a_2^\mathsf{T} \\ \vdots \\ a_m^\mathsf{T} \end{bmatrix}$$

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

$$v_i = \alpha + \beta i T + w_i$$

where $\alpha$ is the sensor bias (which is unknown but the *same* for all sensor measurements), $\beta$ 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 $\alpha$ and the drift term $\beta$ we could just subtract the predictable part of the sensor signal, *i.e.*, $\alpha + \beta i T$ from the sensor signal. But we're interested in the case where we don't know the offset $\alpha$ or the drift coefficient $\beta$. 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.

2. **Navigation from range measurements.** In this problem we are going to study a simple $2D$ navigation system. Let $x \in \mathbb{R}^2$ be the unknown coordinates of a vehicle and let $p_i \in \mathbb{R}^2$ be the known fixed coordinates of beacons for $i = 1, 2, 3, 4$. The vehicle can measure its range or distance $\rho_i \in \mathbb{R}_+$ from the each beacon $i$ and use the four range measurements to estimate its coordinates $x = (x_1, x_2)$.



$\rho \in \mathbb{R}^4_+$ is a nonlinear function of $x \in \mathbb{R}^2$, given by

$$\rho_i(x) = \sqrt{(x_1 - p_{i1})^2 + (x_2 - p_{i2})^2} \qquad i = 1, 2, 3, 4 \tag{1}$$

a) Linearize $\rho(x)$ around $x_0$ and express it in $\delta\rho = A\delta x$ form. Explicitly express the dimension and entries of matrix $A$.

Let $x_0 = (0, 0)$ be the last navigation fix. You would like to estimate the current position $x$ a short time after, based on the changes in the range measurements. However, there is some noise in your range measurements, *i.e.* you have

$$\delta\rho = Ax + v, \tag{2}$$

where the measurement errors $v_i$ are independent, Gaussian, with zero mean and standard deviation 2 (these details are not important for solving the problem). Also note that since $x_0 = (0, 0)$ we have $\delta x = x$. Let the beacon coordinates and range measurements be given by

$$P = \begin{bmatrix} p_1^T \\ p_2^T \\ p_3^T \\ p_4^T \end{bmatrix} = \begin{bmatrix} 64279 & 76604 \\ 113240 & 1976.5 \\ -13691 & -97414 \\ -117390 & 42726 \end{bmatrix}, \qquad \delta\rho = \begin{bmatrix} -9.897 \\ -4.709 \\ 13.062 \\ -0.091 \end{bmatrix} \tag{3}$$

b) Compute the estimated position $\hat{x}_{jem}$ using just enough measurements. In this method you should ignore the last two range measurements and compute the position by inverting only the top $2 \times 2$ half of $A$. Show that this method gives in fact an unbiased estimator of the actual position $x$.

c) Compute the estimated position $\hat{x}_{ls}$ using least-squares.

d) If the actual position is $x = (4.39, 11.13)$, compare the norm of the errors for $\hat{x}_{ls}$ and $\hat{x}_{jem}$. Which method gives a more accurate estimate of the position in this case? Is it true that one of the two methods would always have better accuracy than the other? Briefly justify your answer.

**3. Quadratic placement.** We consider an integrated circuit (IC) that contains $N$ cells or modules that are connected by $K$ wires. We model a cell as a single point in $\mathbb{R}^2$ (which gives its location on the IC) and ignore the requirement that the cells must not overlap. The positions of the cells are

$$(x_1, y_1),\ (x_2, y_2), \ldots, (x_N, y_N),$$

*i.e.*, $x_i$ gives the $x$-coordinate of cell $i$, and $y_i$ gives the $y$-coordinate of cell $i$. We have two types of cells: *fixed cells*, whose positions are fixed and given, and *free cells*, whose positions are to be determined. We will take the first $n$ cells, at positions

$$(x_1, y_1), \ldots, (x_n, y_n),$$

to be the free ones, and the remaining $N - n$ cells, at positions

$$(x_{n+1}, y_{n+1}), \ldots, (x_N, y_N),$$

to be the fixed ones. The task of finding good positions for the free cells is called *placement*. (The fixed cells correspond to cells that are already placed, or external pins on the IC.) There are $K$ wires that connect pairs of the cells. We will assign an orientation to each wire (even though wires are physically symmetric). Specifically, wire $k$ goes *from* cell $I(k)$ *to* cell $J(k)$. Here $I$ and $J$ are functions that map wire number (*i.e.*, $k$) into the origination cell number (*i.e.*, $I(k)$), and the destination cell number (*i.e.*, $J(k)$), respectively. To describe the wire/cell topology and the functions $I$ and $J$, we'll use the *node incidence matrix* $A$ for the associated directed graph. The node incidence matrix $A \in \mathbb{R}^{K \times N}$ is defined as

$$A_{kj} = \begin{cases} 1 & \text{wire } k \text{ goes to cell } j,\ i.e.,\ j = J(k) \\ -1 & \text{wire } k \text{ goes from cell } j,\ i.e.,\ j = I(k) \\ 0 & \text{otherwise.} \end{cases}$$

Note that the $k$th row of $A$ is associated with the $k$th wire, and the $j$th column of $A$ is associated with the $j$th cell. The goal in placing the free cells is to use the smallest amount of interconnect wire, assuming that the wires are run as straight lines between the cells. (In fact, the wires in an IC are *not* run on straight lines directly between the cells, but that's another story. Pretending that the wires do run on straight lines seems to give good placements.) One common method, called *quadratic placement*, is to place the free cells in order to minimize the the total square wire length, given by

$$J = \sum_{k=1}^{K} \left( (x_{I(k)} - x_{J(k)})^2 + (y_{I(k)} - y_{J(k)})^2 \right).$$

a) Explain how to find the positions of the free cells, *i.e.*,

$$(x_1, y_1), \ldots, (x_n, y_n),$$

that minimize the total square wire length. You may make an assumption about the rank of one or more matrices that arise.

b) In this part you will determine the optimal quadratic placement for a specific set of cells and interconnect topology. The mfile `qplace_data.m` defines an instance of the quadratic placement problem. Specifically, it defines the dimensions $n$, $N$, and $K$, and $N-n$ vectors `xfixed` and `yfixed`, which give the $x$- and $y$-coordinates of the fixed cells. The mfile also defines the node incidence matrix `A`, which is $K \times N$. Be sure to explain how you solve this problem, and to explain the matlab source code that solves it (which you must submit). Give the optimal locations of the free cells. Check your placement against various others, such as placing all free cells at the origin. You will also find an mfile that plots a proposed placement in a nice way:
`view_layout(xfree,yfree,xfixed,yfixed,A)`.
This mfile takes as argument the $x$- and $y$-coordinates of the free and fixed cells, as well as the node incidence matrix that describes the wires. It plots the proposed placement. Plot your optimal placement using `view_layout`.

**4. Fleet modeling.** In this problem, we will consider model estimation for vehicles in a fleet. We collect input and output data at multiple time instances, for each vehicle in a fleet of vehicles:

$$x^{(k)}(t) \in \mathbb{R}^n, \quad y^{(k)}(t) \in \mathbb{R}, \quad t = 1, \ldots, T, \quad k = 1, \ldots, K.$$

Here $k$ denotes the vehicle number, $t$ denotes the time, $x^{(k)}(t) \in \mathbb{R}^n$ the input, and $y^{(k)}(t) \in \mathbb{R}$ the output. (In the general case the output would also be a vector; but for simplicity here we consider the scalar output case.)

While it does not affect the problem, we describe a more specific application, where the vehicles are airplanes. The components of the inputs might be, for example, the deflections of various control surfaces and the thrust of the engines; the output might be vertical acceleration. Airlines are required to collect this data, called FOQA data, for every commercial flight. (This description is not needed to solve the problem.)

We will fit a model of the form

$$y^{(k)}(t) \approx a^{\mathsf{T}} x^{(k)}(t) + b^{(k)},$$

where $a \in \mathbb{R}^n$ is the (common) linear model parameter, and $b^{(k)} \in \mathbb{R}$ is the (individual) offset for the $k$th vehicle.

We will choose these to minimize the mean square error

$$E = \frac{1}{TK} \sum_{t=1}^{\mathsf{T}} \sum_{k=1}^{K} \left( y^{(k)}(t) - a^{\mathsf{T}} x^{(k)}(t) - b^{(k)} \right)^2.$$

a) Explain how to find the model parameters $a$ and $b^{(1)}, \ldots, b^{(K)}$.

b) Carry out your method on the data given in `fleet_mod_data.m`. The data is given using cell arrays `X` and `y`. The columns of the $n \times T$ matrix `X{k}` are $x^{(k)}(1), \ldots, x^{(k)}(T)$, and the $1 \times T$ row vector `y{k}` contains $y^{(k)}(1), \ldots, y^{(k)}(T)$. Give the model parameters $a$ and $b^{(1)}, \ldots, b^{(K)}$, and report the associated mean square error $E$. Compare $E$ to the (minimum) mean square error $E^{\text{com}}$ obtained using a common offset $b = b^{(1)} = \cdots = b^{(K)}$ for all vehicles.

By examining the offsets for the different vehicles, suggest a vehicle you might want to have a maintenance crew check out. (This is a simple, straightforward question; we don't want to hear a long explanation or discussion.)

5. **Fitting a Gaussian function to data.** A Gaussian function has the form

$$f(t) = ae^{-(t-\mu)^2/\sigma^2}.$$

Here $t \in \mathbb{R}$ is the independent variable, and $a \in \mathbb{R}$, $\mu \in \mathbb{R}$, and $\sigma \in \mathbb{R}$ are parameters that affect its shape. The parameter $a$ is called the *amplitude* of the Gaussian, $\mu$ is called its *center*, and $\sigma$ is called the *spread* or *width*. We can always take $\sigma > 0$. For convenience we define $p \in \mathbb{R}^3$ as the vector of the parameters, *i.e.*, $p = [a\ \mu\ \sigma]^\mathsf{T}$. We are given a set of data,

$$t_1, \dots, t_N, \qquad y_1, \dots, y_N,$$

and our goal is to fit a Gaussian function to the data. We will measure the quality of the fit by the root-mean-square (RMS) fitting error, given by

$$E = \left( \frac{1}{N} \sum_{i=1}^{N} (f(t_i) - y_i)^2 \right)^{1/2}.$$

Note that $E$ is a function of the parameters $a$, $\mu$, $\sigma$, *i.e.*, $p$. Your job is to choose these parameters to minimize $E$. You'll use the Gauss-Newton method.

a) Work out the details of the Gauss-Newton method for this fitting problem. Explicitly describe the Gauss-Newton steps, including the matrices and vectors that come up. You can use the notation $\Delta p^{(k)} = [\Delta a^{(k)}\ \Delta\mu^{(k)}\ \Delta\sigma^{(k)}]^\mathsf{T}$ to denote the update to the parameters, *i.e.*,

$$p^{(k+1)} = p^{(k)} + \Delta p^{(k)}.$$

(Here $k$ denotes the $k$th iteration.)

b) Get the data $t$, $y$ (and $N$) from the file `gauss_fit_data.m`, available on the class website. Implement the Gauss-Newton (as outlined in part (a) above). You'll need an initial guess for the parameters. You can visually estimate them (giving a short justification), or estimate them by any other method (but you must explain your method). Plot the RMS error $E$ as a function of the iteration number. (You should plot enough iterations to convince yourself that the algorithm has nearly converged.) Plot the final Gaussian function obtained along with the data on the same plot. Repeat for another reasonable, but different initial guess for the parameters. Repeat for another set of parameters that is *not* reasonable, *i.e.*, not a good guess for the parameters. (It's possible, of course, that the Gauss-Newton algorithm doesn't converge, or fails at some step; if this occurs, say so.) Briefly comment on the results you obtain in the three cases.

6. **Smallest input that drives a system to a desired steady-state output.** We start with the discrete-time model of the system used in lecture 1:

$$x(t+1) = A_d x(t) + B_d u(t), \quad y(t) = C_d x(t), \quad t = 1, 2, \dots,$$

where $A_d \in \mathbb{R}^{16\times 16}$, $B_d \in \mathbb{R}^{16\times 2}$, $C_d \in \mathbb{R}^{2\times 16}$. The system starts from the zero state, *i.e.*, $x(1) = 0$. (We start from initial time $t = 1$ rather than the more conventional $t = 0$ since matlab indexes vectors starting from 1, not 0.) The data for this problem can be found in `ss_small_input_data.m`.

The goal is to find an input $u$ that results in $y(t) \to y_{\rm des} = (1, -2)$ as $t \to \infty$ (*i.e.*, asymptotic convergence to a desired output) or, even better, an input $u$ that results in $y(t) = y_{\rm des}$ for $t = T + 1, \ldots$ (*i.e.*, exact convergence after $T$ steps).

a) *Steady-state analysis for desired constant output.* Suppose that the system is in steady-state, *i.e.*, $x(t) = x_{\rm ss}$, $u(t) = u_{\rm ss}$ and $y(t) = y_{\rm des}$ are constant (do not depend on $t$). Find $u_{\rm ss}$ and $x_{\rm ss}$.

b) *Simple simulation.* Find $y(t)$, with initial state $x(1) = 0$, with $u(t) = u_{\rm ss}$, for $t = 1, \ldots, 20000$. Plot $u$ and $y$ versus $t$. If you've done everything right, you should observe that $y(t)$ appears to be converging to $y_{\rm des}$.

You can use the following matlab code to obtain plots that look like the ones in lecture 1.

```
figure;
subplot(411); plot(u(1,:));
subplot(412); plot(u(2,:));
subplot(413); plot(y(1,:));
subplot(414); plot(y(2,:));
```

Here we assume that `u` and `y` are $2 \times 20000$ matrices. There will be two differences between these plots and those in lecture 1: These plots start from $t = 1$, and the plots in lecture 1 scale $t$ by a factor of 0.1.

c) *Smallest input.* Let $u^\star(t)$, for $t = 1, \ldots, T$, be the input with minimum RMS value

$$\left(\frac{1}{T}\sum_{t=1}^{\mathsf{T}} \|u(t)\|^2\right)^{1/2}$$

that yields $x(T + 1) = x_{\rm ss}$ (the value found in part (a)). Note that if $u(t) = u^\star(t)$ for $t = 1, \ldots, T$, and then $u(t) = u_{\rm ss}$ for $t = T + 1, T + 2, \ldots$, then $y(t) = y_{\rm des}$ for $t \geq T + 1$. In other words, we have exact convergence to the desired output in $T$ steps.

For the three cases $T = 100$, $T = 200$, and $T = 500$, find $u^\star$ and its associated RMS value. For each of these three cases, plot $u$ and $y$ versus $t$.

d) Plot the RMS value of $u^\star$ versus $T$ for $T$ between 100 and 1000 (for multiples of 10, if you like). The plot is probably better viewed on a log-log scale, which can be done using the command `loglog` instead of the command `plot`.

**7. Smooth and least-norm force profiles.** Consider the mass/force example described in the lecture notes (slide 5-5) with $n = 10$. For this problem, we are interested in input force sequences which move the mass from an initial position and velocity of zero to final position 1 and final velocity zero.

a) Find the sequence of forces that will move the mass as required, while minimizing the norm of the force vector.

b) Define the *roughness* $R$ of a vector $x \in \mathbb{R}^n$ as

$$R = \sum_{i=0}^{n} (x_{i+1} - x_i)^2,$$

where we let $x_0 = x_{n+1} = 0$. Find the sequence of forces with the smallest roughness $R$. Show both force profiles in a single plot.

*Remark.* Please solve these problems exactly, *i.e.*, do not solve a regularized least-squares problem with $\mu$ set very large or small.