

Exercises

Exercise 2.1 Least Squares Fit of an Ellipse

Suppose a particular object is modeled as moving in an elliptical orbit centered at the origin. Its nominal trajectory is described in rectangular coordinates (r, s) by the constraint equation $x_1 r^2 + x_2 s^2 + x_3 rs = 1$, where x_1 , x_2 , and x_3 are unknown parameters that specify the orbit. We have available the following noisy measurements of the object's coordinates (r, s) at ten different points on its orbit:

(0.6728, 0.0589) (0.3380, 0.4093) (0.2510, 0.3559) (-0.0684, 0.5449)
(-0.4329, 0.3657) (-0.6921, 0.0252) (-0.3681, -0.2020) (0.0019, -0.3769)
(0.0825, -0.3508) (0.5294, -0.2918)

The ten measurements are believed to be equally reliable. For your convenience, these ten pairs of measured (r, s) values have been stored in column vectors named r and s that you can access through the 6.241 locker on Athena. After `add 6.241`, and once in the directory in which you are running Matlab, you can copy the data using `cp /mit/6.241/Public/fall95/hw1rs.mat hw1rs.mat`. Then, in Matlab, type `load hw1rs` to load the desired data; type `who` to confirm that the vectors r and s are indeed available.

Using the assumed constraint equation, we can arrange the given information in the form of the linear system of (approximate) equations $Ax \approx b$, where A is a known 10×3 matrix, b is a known 10×1 vector, and $x = (x_1, x_2, x_3)^T$. This system of 10 equations in 3 unknowns is inconsistent. We wish to find the solution x that minimizes the Euclidean norm (or length) of the error $Ax - b$. Compare the solutions obtained by using the following four Matlab invocations, each of which in principle gives the desired least-square-error solution:

- (a) $x = A \backslash b$
- (b) $x = \text{pinv}(A) * b$
- (c) $x = \text{inv}(A' * A) * A' * b$
- (d) $[q, r] = \text{qr}(A)$, followed by implementation of the approach described in Exercise 3.1.

For more information on these commands, try `help slash`, `help qr`, `help pinv`, `help inv`, etc. [Incidentally, the prime, $'$, in Matlab takes the transpose of the *complex conjugate* of a matrix; if you want the ordinary transpose of a complex matrix C , you have to write C' or `transp(C)`.]

You should include in your solutions a plot the ellipse that corresponds to your estimate of x . If you create the following function file in your Matlab directory, with the name `ellipse.m`, you can obtain the polar coordinates `theta`, `rho` of n points on the ellipse specified by the parameter vector x . To do this, enter `[theta,rho]=ellipse(x,n)`; at the Matlab prompt. You can then plot the ellipse by using the `polar(theta,rho)` command.

```
function [theta,rho]=ellipse(x,n)
% [theta,rho]=ellipse(x,n)
%
% The vector x = [x(1),x(2),x(3)]', defines an ellipse centered at the origin
% via the equation x(1)*r^2 + x(2)*s^2 + x(3)*r*s = 1.
% This routine generates the polar coordinates of points on the ellipse,
% to send to a plot command. It does this by solving for the radial
% distance in n equally spaced angular directions.
% Use polar(theta,rho) to actually plot the ellipse.
```

```
theta = 0:(2*pi/n):(2*pi);
a = x(1)*cos(theta).^ 2 + x(2)*sin(theta).^ 2 + x(3)*(cos(theta).*sin(theta));
rho = ones(size(a))./sqrt(a);
```

Exercise 2.2 Approximation by a Polynomial

Let $f(t) = 0.5e^{0.8t}$, $t \in [0, 2]$.

- (a) Suppose 16 exact measurements of $f(t)$ are available to you, taken at the times t_i listed in the array T below:

$$T = \begin{bmatrix} 2 \cdot 10^{-3}, & 0.136, & 0.268, & 0.402, & 0.536, & 0.668, & 0.802, & 0.936, \\ 1.068, & 1.202, & 1.336, & 1.468, & 1.602, & 1.736, & 1.868, & 2.000 \end{bmatrix}$$

Use Matlab to generate these measurements:

$$y_i = f(t_i) \quad i = 1, \dots, 16 \quad t_i \in T$$

Now determine the coefficients of the least square error polynomial approximation of the measurements, for

1. a polynomial of degree 15, $p_{15}(t)$;
2. a polynomial of degree 2, $p_2(t)$.

Compare the quality of the two approximations by plotting $y(t_i)$, $p_{15}(t_i)$ and $p_2(t_i)$ for all t_i in T . To see how well we are approximating the function on the whole interval, also plot $f(t)$, $p_{15}(t)$ and $p_2(t)$ on the interval $[0, 2]$. (Pick a very fine grid for the interval, e.g. $t=[0:1000]'/500$.) Report your observations and comments.

- (b) Now suppose that your measurements are affected by some noise. Generate the measurements using

$$y_i = f(t_i) + e(t_i) \quad i = 1, \dots, 16 \quad t_i \in T$$

where the vector of noise values can be generated in the following way:

```
randn('seed', 0);
e = randn(size(T));
```

Again determine the coefficients of the least square error polynomial approximation of the measurements for

1. a polynomial of degree 15, $p_{15}(t)$;
2. a polynomial of degree 2, $p_2(t)$.

Compare the two approximations as in part (a). Report your observations and comments. Explain any surprising results.

- (c) So far we have obtained polynomial approximations of $f(t)$, $t \in [0, 2]$, by approximating the measurements at $t_i \in T$. We are now interested in minimizing the square error of the polynomial approximation over the whole interval $[0, 2]$:

$$\min \|f(t) - p_n(t)\|_2^2 = \min \int_0^2 |f(t) - p_n(t)|^2 dt$$

where $p_n(t)$ is some polynomial of degree n . Find the polynomial $p_2(t)$ of degree 2 that solves the above problem. Are the optimal $p_2(t)$ in this case and the optimal $p_2(t)$ of parts (a) and (b) very different from each other? Elaborate.

Exercise 3.1 Least Square Error Solution We begin with a mini-tutorial on orthogonal and unitary matrices. An *orthogonal matrix* may be defined as a *square* real matrix whose columns are of unit length and mutually orthogonal to each other — i.e., its columns form an *orthonormal* set. It follows quite easily (as you should try and verify for yourself) that:

- the *inverse* of an orthogonal matrix is just its transpose;
- the *rows* of an orthogonal matrix form an orthonormal set as well;
- the usual Euclidean *inner product* of two real vectors v and w , namely the scalar $v'w$, equals the inner product of Uv and Uw , if U is an orthogonal matrix — and therefore the length of v , namely $\sqrt{v'v}$, equals that of Uv .

A *unitary matrix* is similarly defined, except that its entries are allowed to be complex — so its inverse is the *complex conjugate* of its transpose. A fact about orthogonal matrices that turns out to be important in several numerical algorithms is the following: Given a real $m \times n$ matrix A of full column rank, it is possible (in many ways) to find an orthogonal matrix U such that

$$UA = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

where R is a nonsingular, upper-triangular matrix. (If A is *complex*, then we can find a *unitary* matrix U that leads to the same equation.) To see how to compute U in Matlab, read the comments obtained by typing `help qr`; the matrix Q that is referred to in the comments is just U' .

We now turn to the problem of interest. Given a real $m \times n$ matrix A of full column rank, and a real m -vector y , we wish to approximately satisfy the equation $y = Ax$. Specifically, let us choose the vector x to minimize $\|y - Ax\|^2 = (y - Ax)'(y - Ax)$, the squared Euclidean length of the “error” $y - Ax$. By invoking the above results on orthogonal matrices, **show that** (in the notation introduced earlier) the minimizing x is

$$\hat{x} = R^{-1}y_1$$

where y_1 denotes the vector formed from the first n components of Uy . (In practice, we would not bother to find R^{-1} explicitly. Instead, taking advantage of the upper-triangular structure of R , we would solve the system of equations $R\hat{x} = y_1$ by back substitution, starting from the last equation.)

The above way of solving a least-squares problem (proposed by Householder in 1958, but sometimes referred to as Golub's algorithm) is numerically preferable in most cases to solving the "normal equations" in the form $\hat{x} = (A'A)^{-1}A'y$, and is essentially what Matlab does when you write $\hat{x} = A \backslash y$. An (oversimplified!) explanation of the trouble with the normal equation solution is that it implicitly evaluates the product $(R'R)^{-1}R'$, whereas the Householder/Golub method recognizes that this product simply equals R^{-1} , and thereby avoids unnecessary and error prone steps.

Exercise 3.2 Suppose the input sequence $\{u_j\}$ and the output sequence $\{y_j\}$ of a particular system are related by

$$y_k = \sum_{i=1}^n h_i u_{k-i}$$

where all quantities are scalar.

(i) Assume we want to have y_n equal to some specified number $\bar{y}$. Determine $u_0, \dots, u_{n-1}$ so as to achieve this while minimizing $u_0^2 + \dots + u_{n-1}^2$.

(ii) Suppose now that we are willing to relax our objective of exactly attaining $y_n = \bar{y}$. This leads us to the following modified problem. Determine $u_0, \dots, u_{n-1}$ so as to minimize

$$r(\bar{y} - y_n)^2 + u_0^2 + \dots + u_{n-1}^2$$

where r is a positive weighting parameter.

(a) Solve the modified problem.

(b) What do you expect the answer to be in the limiting cases of $r = 0$ and $r = \infty$? Show that your answer in (a) indeed gives you these expected limiting results.

Exercise 3.3 Return to the problem considered in Example 3.4. Suppose that, in addition to requiring $p(T) = y$ for a specified y , we also want $\dot{p}(T) = 0$. In other words, we want to bring the mass *to rest* at the position y at time T . Of all the force functions $x(t)$ that can accomplish this, determine the one that minimizes $\langle x(t), x(t) \rangle = \int_0^T x^2(t) dt$.